



WPS Reference for Language Elements

Contents

How to read railroad syntax diagrams.....	13
--	-----------

Common Syntax.....	15
---------------------------	-----------

Identifying access and view descriptors.....	15
Access Descriptors.....	15
View Descriptors.....	15
Identifying Views.....	15
Identifying datasets.....	16
Dataset.....	16
Input dataset.....	16
Output dataset.....	16
Dataset options.....	17
Expressions.....	24
SAS Language expressions.....	24
SQL expressions.....	28
External Files.....	29
LOCALE Values.....	29
Variable Lists.....	32
ENCODING Values.....	32
z/OS encoding values.....	32
Encoding values on non-z/OS platforms.....	34
Not supported for server encoding.....	36
LOCALE Values.....	37

WPS Core.....	42
----------------------	-----------

System options.....	42
Setting system options.....	42
Displaying system options.....	45
Restricting system options.....	46
COMMUNICATIONS group system options.....	48
DATABASE ENGINE group system options.....	71
DB2 group system options.....	78
EMAIL group system options.....	81
ENVDISPLAY group system options.....	87
ENVFILES group system options.....	93
ERRORHANDLING group system options.....	116
EXECMODES group system options.....	125
EXTFILES group system options.....	134
FLE_CONTROL group system options.....	156
GRAPHICS group system options.....	159

HUB group system options.....	161
IMS group system options.....	165
INPUTCONTROL group system options.....	166
INSTALL group system options.....	180
ISPF group system options.....	182
LANGUAGECONTROL group system options.....	196
LOG_LISTCONTROL group system options.....	207
LISTCONTROL group system options.....	215
LOGCONTROL group system options.....	231
MACRO group system options.....	264
MEMORY group system options.....	285
ODSPRINT group system options.....	291
PERFORMANCE group system options.....	297
SASFILES group system options.....	309
SMF group system options.....	348
SORT group system options.....	349
SQL group system options.....	368
TLS group system options.....	371
Formats.....	373
Core formats.....	374
Basic character formats.....	378
Bidirectional formats.....	384
Unicode formats.....	387
Simple numeric formats.....	397
Numeric date formats.....	428
ISO8601 date formats.....	476
International date formats.....	491
NLS-sensitive date formats.....	511
NLS-sensitive money formats.....	526
NLS-sensitive numeric formats.....	532
Informats.....	538
Core informats.....	540
Basic character informats.....	541
Bidirectional informats.....	548
Unicode informats.....	550
Simple numeric informats.....	558
Numeric date informats.....	575
ISO8601 date informats.....	587
NLS-sensitive date informats.....	598
NLS-sensitive money informats.....	600
NLS-sensitive numeric informats.....	603
Global statements.....	605
Comment.....	606
CATNAME.....	606
DSNEXST.....	606
ENDSAS.....	606

ENDWPS.....	606
FILENAME statements.....	607
FOOTNOTE.....	616
%INCLUDE.....	618
LIBNAME.....	618
MISSING.....	619
OPTIONS.....	619
PAGE.....	621
RUN.....	621
SKIP.....	621
SYSTASK statements.....	621
TITLE.....	622
WAITFOR.....	624
X statements.....	624
DATA step statements.....	625
NEW.....	625
ABORT.....	625
ARRAY.....	625
ATTRIB.....	626
BY.....	626
CALL.....	626
CONTINUE.....	626
DATA.....	627
DELETE.....	627
DESCRIBE.....	627
DO.....	627
DO, iterative.....	627
DO UNTIL.....	628
DO WHILE.....	628
DROP.....	628
END.....	628
ERROR.....	629
EXECUTE.....	629
FILE.....	629
FORMAT.....	632
GO TO.....	632
IF, subsetting.....	632
IF-THEN/ELSE.....	632
INFILE.....	633
INFORMAT.....	635
INPUT.....	635
KEEP.....	637
LABEL.....	637
Labels, Statement.....	637
LEAVE.....	637
LENGTH.....	637

LINK.....	638
LIST.....	638
MERGE.....	638
MODIFY.....	638
OUTPUT.....	639
PUT.....	639
PUTLOG.....	642
REDIRECT.....	642
REMOVE.....	642
RENAME.....	642
REPLACE.....	643
RETAIN.....	643
RETURN.....	643
SELECT.....	643
SET.....	644
SKIP.....	644
STOP.....	644
Sum.....	644
UPDATE.....	645
WHERE.....	645
Describing data in a DATA step.....	645
DATA step functions and CALL routines.....	646
Array functions.....	648
Bitwise functions.....	652
Combination functions and CALL routines.....	658
Comparison functions.....	671
Cryptographic functions.....	674
Date and time functions and CALL routine.....	684
Dataset input and output functions and CALL routines.....	723
Decision forest functions and CALL routines.....	764
Difference and lag functions.....	768
Distribution-based functions and CALL routines.....	775
External file functions.....	1367
External module functions and CALL routines.....	1420
Financial functions.....	1434
Internet functions.....	1578
List functions and CALL routines.....	1593
ISPF CALL routines.....	1741
Macro functions and CALL routines.....	1743
Mathematical functions and CALL routines.....	1757
Memory manipulation functions.....	1880
Miscellaneous functions.....	1894
National language support functions.....	1899
Regular expression functions and CALL routines.....	1913
Sequence manipulation functions.....	1932
String functions and CALL routines.....	1941

System command function and <code>CALL</code> routine.....	2110
System information functions.....	2113
Truncation and rounding functions.....	2120
Unicode functions.....	2146
Value formatting and assignment functions.....	2151
Variable information functions and <code>CALL</code> routines.....	2163
Web functions.....	2215
Zipcode functions.....	2221
DATA step Components.....	2232
HASH Component.....	2232
HITER Component.....	2237
JAVAOBJ Component.....	2238
Output Delivery System.....	2242
ODS global statements.....	2242
ODS.....	2245
ODS MARKUP.....	2245
ODS EXCEL.....	2254
ODS OLDHTML.....	2257
ODS LISTING.....	2257
ODS PDF.....	2258
Procedures.....	2261
ACCESS procedure.....	2261
APPEND procedure.....	2264
APPSRV Procedure.....	2265
CATALOG procedure.....	2283
CDISC procedure.....	2287
CHART procedure.....	2292
CIMPORT procedure.....	2296
COMPARE procedure.....	2298
CONTENTS procedure.....	2301
COPY procedure.....	2303
CORR procedure.....	2306
CPORT procedure.....	2310
DATASETS procedure.....	2312
DBLOAD procedure.....	2321
DELETE procedure.....	2325
EXPORT procedure.....	2326
FMTLIB procedure.....	2330
FONT Procedure.....	2334
FORMAT procedure.....	2340
FORMS procedure.....	2344
FREQ procedure.....	2346
HADOOP Procedure.....	2356
HTTP procedure.....	2358
IMPORT procedure.....	2363
JSON Procedure.....	2370

JAVAINFO procedure.....	2385
MEANS procedure.....	2389
OPTIONS procedure.....	2396
OPTLOAD procedure.....	2405
OPTSAVE procedure.....	2407
PDS procedure.....	2408
PDSCOPY procedure.....	2410
PLOT procedure.....	2411
PRINT procedure.....	2413
PRINTTO procedure.....	2417
PWENCODE procedure.....	2418
PYTHON procedure.....	2420
R Procedure.....	2436
RANK procedure.....	2456
RELEASE procedure.....	2458
REPORT procedure.....	2460
SOAP procedure.....	2468
SORT procedure.....	2471
SOURCE procedure.....	2473
SQL Procedure.....	2476
STANDARD procedure.....	2495
SUMMARY procedure.....	2497
TABLEAU procedure.....	2504
TABULATE procedure.....	2505
TEMPLATE procedure.....	2512
TRANPOSE procedure.....	2534
TRANTAB procedure.....	2536
UNIVARIATE procedure.....	2539
Library engines.....	2579
CVP.....	2579
JSON.....	2580
SASDASD.....	2580
SAS7BDAT.....	2580
SD2.....	2581
SASSEQ.....	2581
V8SEQ.....	2581
V9SEQ.....	2581
WPDSEQ.....	2581
WPD.....	2581
WPD (z/OS).....	2582
WPD1.....	2583
WPDV2.....	2583
WPDV2 (z/OS).....	2583
XML.....	2584
XPORT.....	2585
Macros.....	2586

Automatic macro variables.....	2586
Macro processor statements.....	2591
Macro processor functions.....	2596

WPS Graphing.....2604

Global statements.....	2604
AXIS.....	2604
GOPTIONS.....	2606
LEGEND.....	2608
PATTERN.....	2610
SYMBOL.....	2610
Graphing procedures.....	2612
GANNNO procedure.....	2612
GBARLINE procedure.....	2613
GCHART procedure.....	2618
GINSIDE procedure.....	2626
GMAP procedure.....	2627
GOPTIONS procedure.....	2630
GPLOT procedure.....	2631
GPROJECT procedure.....	2639
GREduce procedure.....	2641
GREMOVE procedure.....	2643
GREPLAY procedure.....	2644
GSLIDE procedure.....	2646
MAPIMPORT procedure.....	2647
SGPANEL procedure.....	2648
SGPLOT procedure.....	2709
SGSCATTER procedure.....	2777

WPS Statistics.....2789

Statistics procedures.....	2789
ACECLUS procedure.....	2789
ANOVA procedure.....	2792
ASSOCRULES procedure.....	2796
BIN procedure.....	2809
BOXPLOT procedure.....	2812
CANCORR procedure.....	2817
CANDISC procedure.....	2821
CLUSTER procedure.....	2825
CORRESP procedure.....	2829
DISCRIM procedure.....	2835
DISTANCE procedure.....	2840
FACTOR procedure.....	2846
FASTCLUS procedure.....	2851
GAM procedure.....	2855
GENMOD procedure.....	2861

GLM procedure.....	2873
GLMMOD procedure.....	2885
GLMSELECT procedure.....	2888
ICLIFETEST procedure.....	2900
KDE procedure.....	2907
LIFEREG procedure.....	2911
LIFETEST procedure.....	2920
LOESS procedure.....	2926
LOGISTIC procedure.....	2933
MDS procedure.....	2948
MI procedure.....	2952
MIANALYZE procedure.....	2961
MIXED procedure.....	2965
MODECLUS procedure.....	2979
NESTED procedure.....	2982
NLIN procedure.....	2985
NPAR1WAY procedure.....	2995
PHREG procedure.....	3001
PLAN procedure.....	3011
PLS procedure.....	3015
POWER procedure.....	3019
PRINCOMP procedure.....	3072
PROBIT procedure.....	3076
QUANTREG procedure.....	3085
REG procedure.....	3093
ROBUSTREG procedure.....	3104
RSREG procedure.....	3113
SCORE procedure.....	3120
SIMNORMAL procedure.....	3122
STDIZE procedure.....	3125
STEPPDISC procedure.....	3129
SURVEYSELECT procedure.....	3134
TPSPLINE procedure.....	3138
TRANSREG procedure.....	3142
TREE procedure.....	3153
TTEST procedure.....	3157
VARCLUS procedure.....	3161
VARCOMP procedure.....	3166

WPS Machine Learning..... 3169

DECISIONFOREST procedure.....	3170
About decision forests.....	3170
Using the DECISIONFOREST procedure.....	3173
DECISIONFOREST procedure reference.....	3178
DECISIONTREE procedure.....	3188
About decision trees.....	3188

Predictive power criteria.....	3190
Using the DECISIONTREE procedure.....	3193
DECISIONTREE procedure reference.....	3196
GMM procedure.....	3209
About Gaussian mixture models.....	3209
Mixture models.....	3215
Gaussian mixture models.....	3216
Posterior probabilities.....	3216
Maximum likelihood and expectation maximisation.....	3217
Model selection.....	3218
Using the GMM procedure.....	3221
GMM procedure reference.....	3237
GMM bibliography.....	3249
MLP procedure.....	3250
About multilayer perceptrons.....	3250
Specifying network structure.....	3252
Preprocessing data.....	3252
Training a network.....	3253
Loading and saving a trained network.....	3254
Results and generated output.....	3255
Using the MLP procedure.....	3255
MLP procedure reference.....	3262
OPTIMALBIN procedure.....	3296
About optimal binning.....	3296
Predictive power criteria.....	3297
Using the OPTIMALBIN procedure.....	3300
OPTIMALBIN procedure reference.....	3305
SEGMENT procedure.....	3314
About segmentation.....	3314
Segmentation calculations.....	3315
Difference functions.....	3316
Weighting factor.....	3320
ODS Outputs.....	3322
Scoring datasets.....	3322
Saving and reusing segmentation models.....	3323
Using the SEGMENT procedure.....	3323
SEGMENT procedure reference.....	3329
SEGMENT bibliography.....	3348
SVM procedure.....	3349
About support vector machines.....	3349
SVM classification.....	3349
SVM Regression.....	3354
Data standardisation.....	3354
Encoding categorical variables.....	3355
Weighting and standardisation.....	3355
Using the SVM procedure.....	3356

SVM procedure reference.....	3363
SVM bibliography.....	3387
WPS Operational Research.....	3389
Operational research procedures.....	3389
LP procedure.....	3389
WPS Quality Control.....	3394
Quality control procedures.....	3394
CAPABILITY Procedure.....	3394
WPS Timeseries.....	3441
Timeseries procedures.....	3441
ARIMA procedure.....	3441
AUTOREG procedure.....	3449
EXPAND procedure.....	3455
FORECAST procedure.....	3462
LOAN procedure.....	3466
X12 procedure.....	3472
WPS Communicate.....	3482
Global statements.....	3482
ENDRSUBMIT.....	3482
RSUBMIT.....	3482
SIGNOFF.....	3486
SIGNON.....	3487
WAITFOR.....	3491
Macro processor statements.....	3492
%SYSLPUT.....	3492
%SYSRPUT.....	3492
WPS Communicate procedures.....	3493
DOWNLOAD Procedure.....	3493
UPLOAD Procedure.....	3497
Data Engines.....	3502
WPS Engine for Access.....	3502
ACCESS.....	3502
WPS Engine for Actian Matrix.....	3507
ACTIANMATRIX.....	3507
WPS Engine for DB2.....	3514
DB2.....	3514
DB2OLD.....	3528
DB2 (for z/OS).....	3537
DB2OLD (for z/OS).....	3547

DB2EXT Procedure.....	3553
WPS Engine for Excel.....	3554
EXCEL.....	3554
XLSX.....	3558
WPS Engine for Greenplum.....	3561
GREENPLUM.....	3561
WPS Engine for Hadoop.....	3568
HADOOP.....	3568
WPS Engine for Informix.....	3573
INFORMIX.....	3573
WPS Engine for Kognito.....	3578
KOGNITIO.....	3578
WPS Engine for MariaDB.....	3584
MARIADB.....	3584
WPS Engine for MySQL.....	3591
MYSQL.....	3591
WPS Engine for Netezza.....	3598
NETEZZA.....	3598
NETEZZAOLD.....	3610
WPS Engine for ODBC.....	3618
ODBC.....	3618
ODBCOLD.....	3628
WPS Engine for OLEDB.....	3639
OLEDB.....	3639
WPS Engine for Oracle.....	3649
Data Types in Oracle.....	3649
How to use the Oracle engine.....	3651
ORACLE connection reference.....	3652
WPS Engine for PostgreSQL.....	3725
POSTGRESQL.....	3725
WPS Engine for Sand.....	3731
SAND.....	3731
WPS Engine for SQL Server.....	3736
SQLSERVER.....	3736
SQLSERVEROLD.....	3750
WPS Engine for Sybase.....	3763
SYBASE.....	3763
WPS Engine for Sybase IQ.....	3768
SYBASEIQ.....	3768
WPS Engine for Teradata.....	3776
TERADATA.....	3776
WPS Engine for Vertica.....	3782
VERTICA.....	3782

Legal Notices.....	3789
---------------------------	-------------

How to read railroad syntax diagrams


Railroad diagrams are a graphical syntax notation that accompanies significant language structures such as procedures, statements and so on.

The description of each language concept commences with its syntax diagram.

Entering text

Text that should be entered exactly as displayed is shown in a typewriter font :

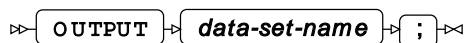


This example describes a fragment of syntax in which the keyword `OUTPUT` is followed by a semi-colon character: `;`. The syntax diagram form is: .

Generally the case of the text is not significant, but in this reference, it is the convention to use upper-case for keywords.

Placeholder items

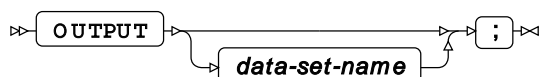
Placeholders that should be substituted with relevant, context-dependent text are rendered in a lower-case, italic font :



Here, the keyword `OUTPUT` should be entered literally, but *data-set-name* should be replaced by something appropriate to the program – in this case, the name of a dataset to add an observation to.

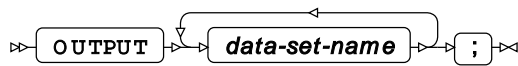
Optionality

When items are optional, they appear on a branch below the main line in railroad diagrams. Optionality is represented by an alternative unimpeded path through the diagram:



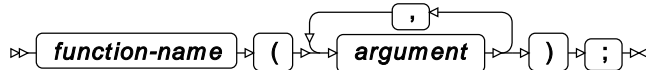
Repetition

In syntax diagrams, repetition is depicted with a return loop that optionally specifies the separator that should be placed between multiple instances.



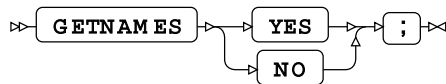
Above, the keyword `OUTPUT` should be entered literally, and it should be followed by one or more repetitions of *data-set-name* - in this case, no separator other than a space has been asked for.

The example below shows the use of a separator.



Choices

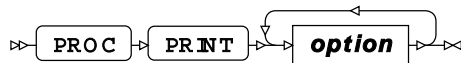
In syntax diagrams, the choice is shown by several parallel branches.



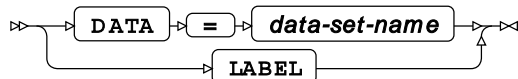
In the above example, the keyword `GETNAMES` should be entered literally, and then either the keyword `YES` or the keyword `NO`.

Fragments

When the syntax is too complicated to fit in one definition, it might be broken into fragments:



option



Above, the whole syntax is split into separate syntax diagram fragments. The first indicates that `PROC PRINT` should be followed by one or more instances of an *option*, each of which must adhere to the syntax given in the second diagram.

Common Syntax

Many language elements in this manual refer to common syntax constructs such as expressions, datasets or variable lists. They are detailed below.

Identifying access and view descriptors

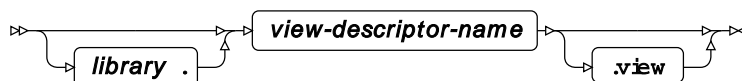
Access Descriptors

Access descriptor

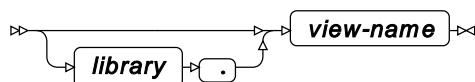


View Descriptors

View descriptor



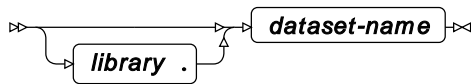
Identifying Views



Identifying datasets

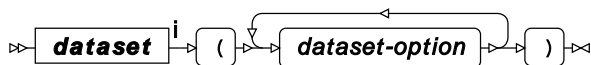
Dataset

Dataset



Input dataset

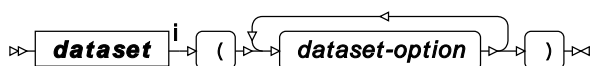
Input dataset with options



ⁱ See [Dataset](#) (page 16).

Output dataset

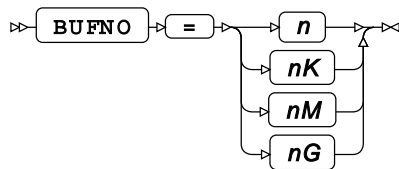
Output dataset with options



ⁱ See [Dataset](#) (page 16).

Dataset options

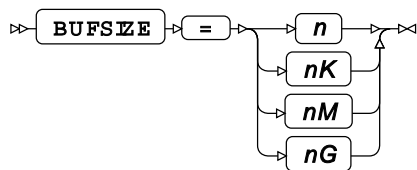
BUFNO



Valid for:

- Input: yes
- Output: no

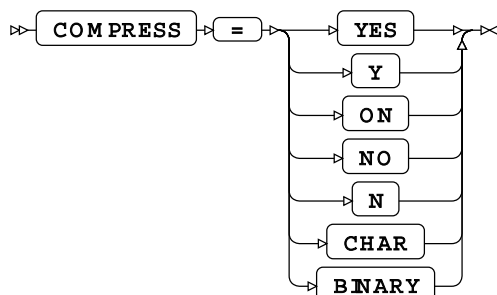
BUFSIZE



Valid for:

- Input: yes
- Output: no

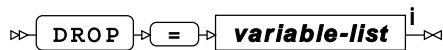
COMPRESS



Valid for:

- Input: yes
- Output: yes

DROP



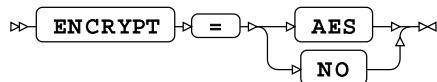
ⁱ See [Variable Lists](#) (page 32).

Valid for:

- Input: yes
- Output: yes

ENCRYPT

Specifies that dataset is encrypted.



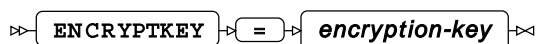
Valid for:

- Input: yes
- Output: yes

The `ENCRYPT` dataset option specifies the encryption mechanism, using the key defined in the `ENCRYPTKEY` dataset option. The supported encryption mechanism is Advanced Encryption Standard (AES)

ENCRYPTKEY

Specifies the key used to encode the dataset.



Valid for:

- Input: yes
- Output: yes

The specified *encryption-key* is used by the `ENCRYPT` dataset option to encrypt and decrypt the dataset.

The *encryption-key* is encoded using the current system session encoding. You will therefore not be able to use an encrypted dataset in a session that uses a different encoding if the key contains characters whose representation differs between the session encodings.

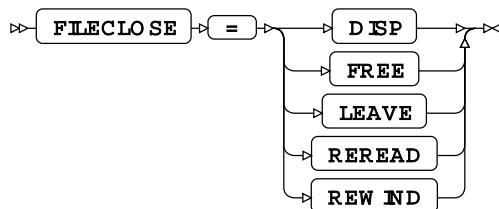
If you need to transfer a dataset between machines with different session encoding, we strongly recommend you use a hexadecimal-encoded *encryption-key* to ensure the key is the same in all session encodings you need to use.

Warning:

If the *encryption-key* is lost you will be unable to recover encrypted data.

You must therefore ensure the specified *encryption-key* is securely stored. WPS does not record the encryption key and is unable to recover a key that has been lost.

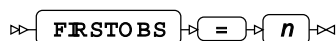
FILECLOSE



Valid for:

- Input: yes
- Output: no

FIRSTOBS

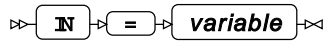


Valid for:

- Input: yes

- Output: no

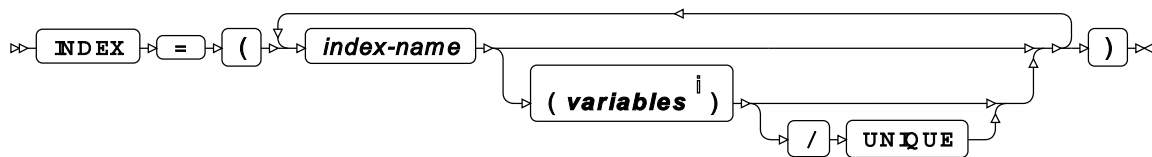
IN



Valid for:

- Input: yes
- Output: no

INDEX

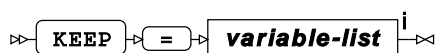


ⁱ See [Variable Lists](#) (page 32).

Valid for:

- Input: no
- Output: yes

KEEP

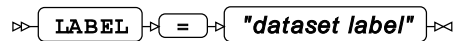


ⁱ See [Variable Lists](#) (page 32).

Valid for:

- Input: yes
- Output: yes

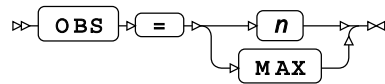
LABEL



Valid for:

- Input: no
- Output: yes

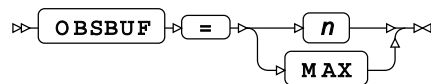
OBS



Valid for:

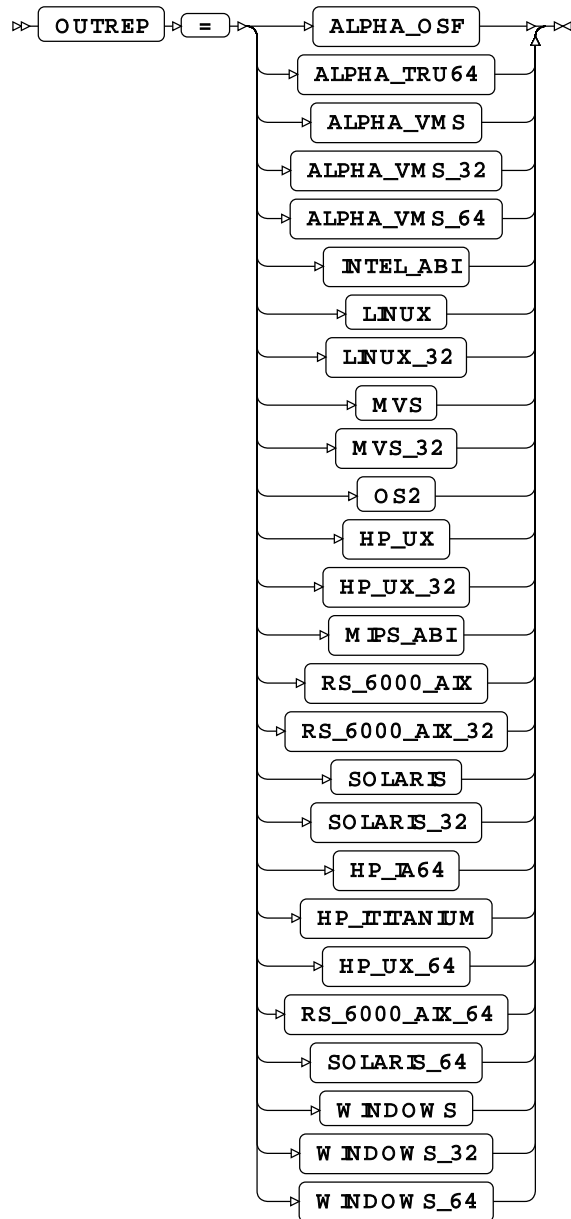
- Input: yes
- Output: no

OBSBUF



This dataset option is only valid for views and determines the number of observations that can be buffered when executing a view in parallel. If specified, it overrides the `VBUF SIZE` system option.

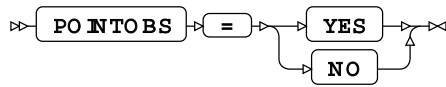
OUTREP



Valid for:

- Input: yes
- Output: no

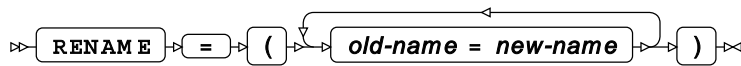
POINTOBS



Valid for:

- Input: yes
- Output: yes

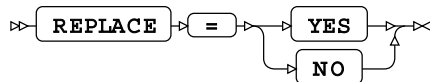
RENAME



Valid for:

- Input: yes
- Output: yes

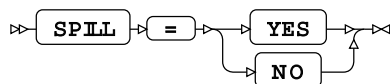
REPLACE



Valid for:

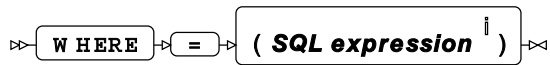
- Input: no
- Output: yes

SPILL



This dataset option is only valid for views and specifies if the view gets spilled to disk or executed in parallel.

WHERE



ⁱ See *SQL expressions* [↗](#) (page 28).

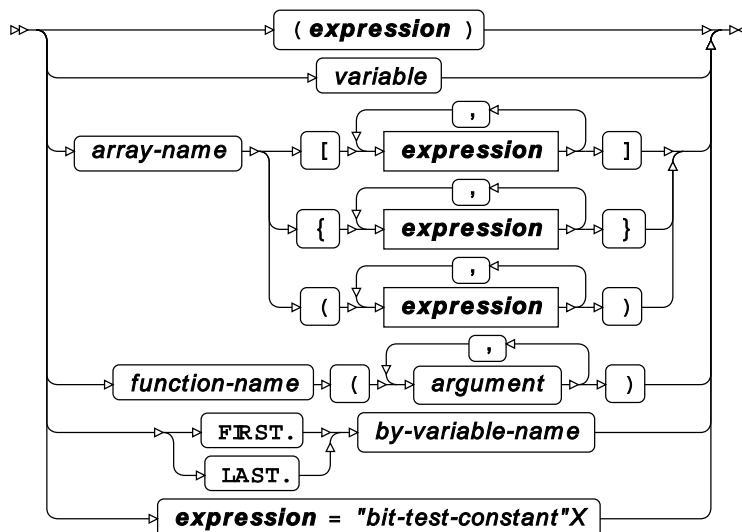
Valid for:

- Input: yes
- Output: no

Expressions

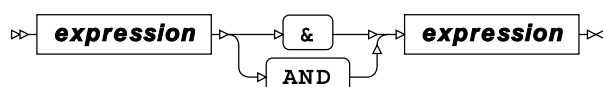
SAS Language expressions

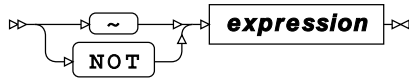
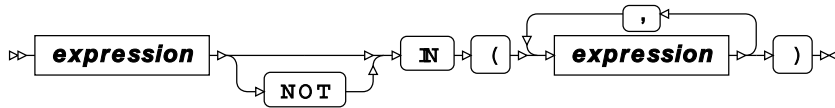
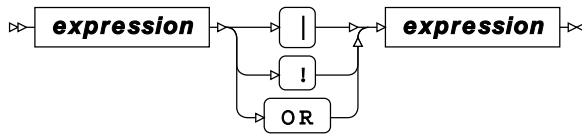
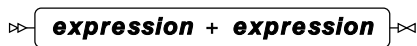
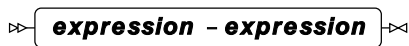
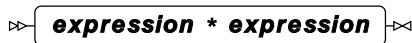
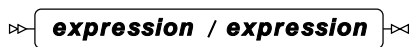
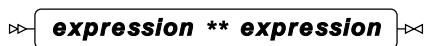
Expression



Logical operators

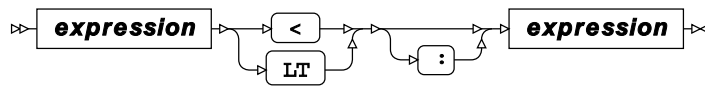
And operator



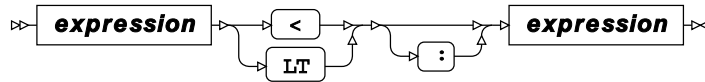
NOT operator**IN operator****OR operator****Arithmetic operators****Addition operator****Subtraction operator****Multiplication operator****Division operator****Power operator****Unary negation operator**

Comparison operators

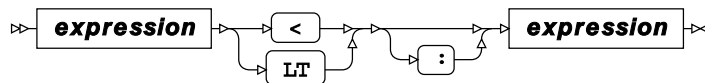
Choose minimum



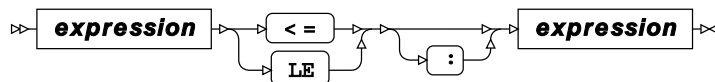
Less-than operator



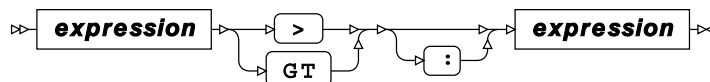
Less-than operator



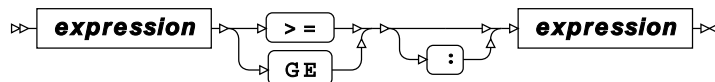
Less-than or equal-to operator



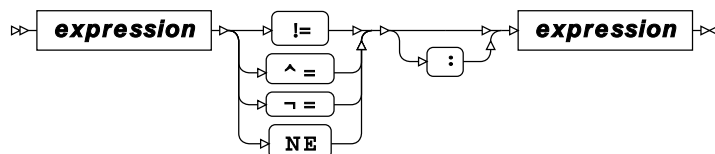
Greater-than or equal-to operator



Greater-than or equal-to operator

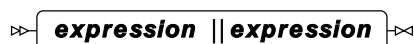


Not-equal-to operator



String concatenation

Concatenation operator



Numeric comparison

Select Maximum

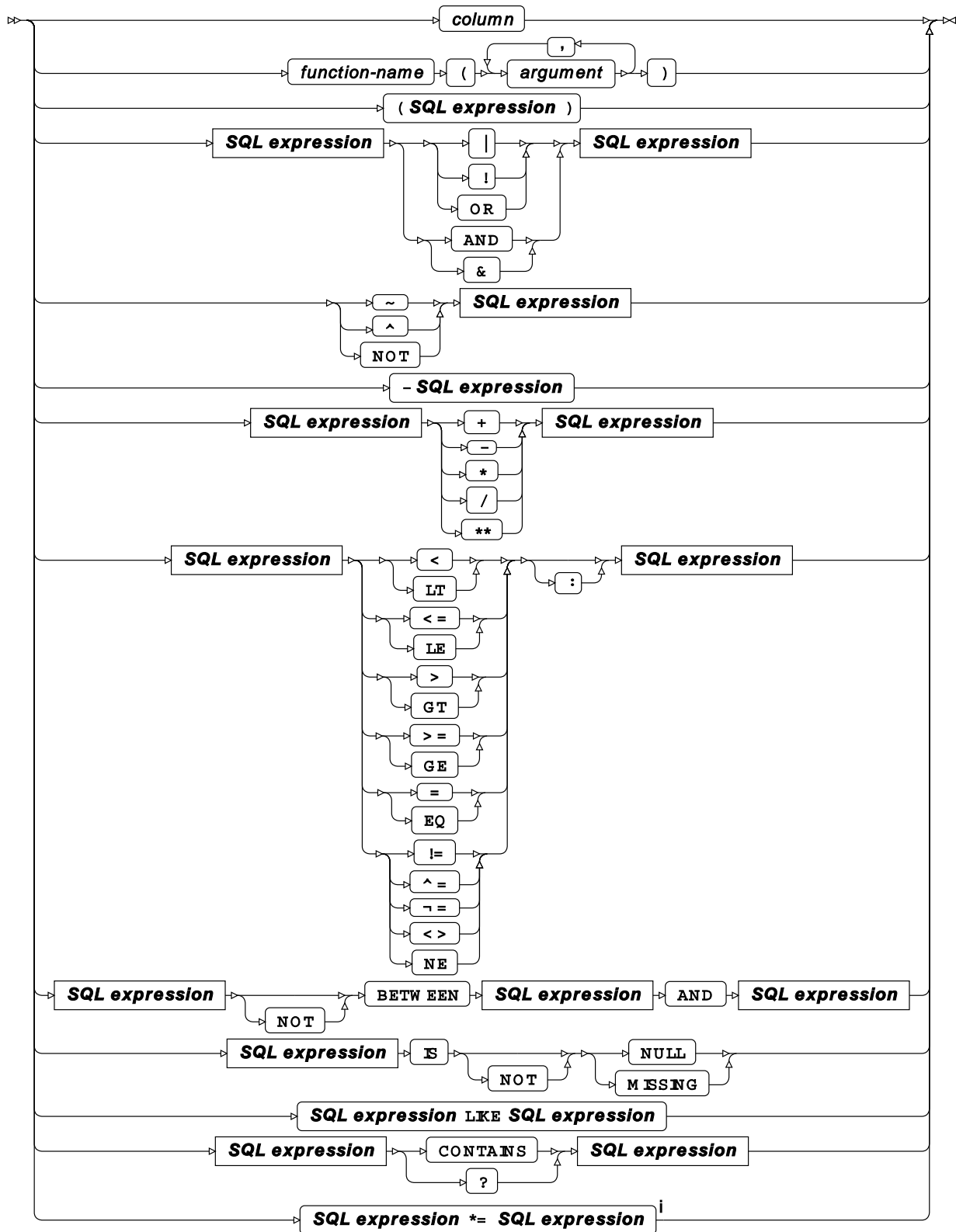
▷ ***expression < > expression*** ◁

Select minimum

▷ ***expression > < expression*** ◁

SQL expressions

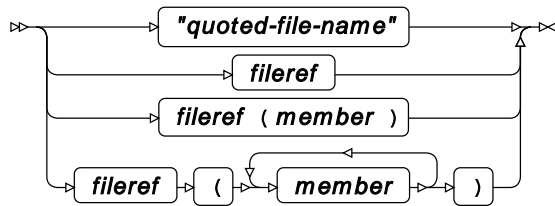
SQL expression



ⁱ Sounds like.

External Files

External files can be referenced in a number of ways.



Not all of the ways of referring to external files are valid in all circumstances. In particular, when referring to output files, at least one file must be identified.

LOCALE Values

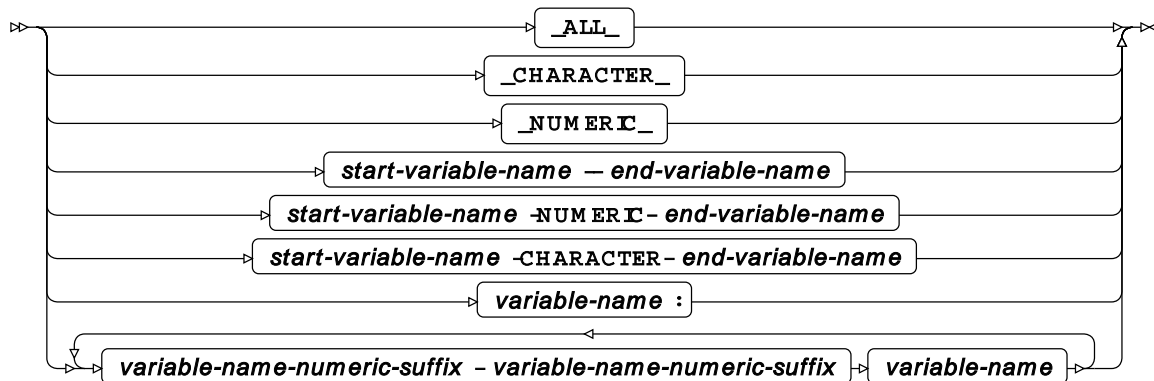
Supported LOCALE Values:

- Arabic_Algeria
- Arabic_Bahrain
- Arabic_Egypt
- Arabic_Jordan
- Arabic_Kuwait
- Arabic_Lebanon
- Arabic_Morocco
- Arabic_Oman
- Arabic_Qatar
- Arabic_SaudiArabia
- Arabic_Tunisia
- Arabic_UnitedArabEmirates
- Bulgarian_Bulgaria
- Byelorussian_Belarus
- Croatian_Croatia
- Czech_CzechRepublic
- Danish_Denmark

- Dutch_Belgium
- Dutch_Netherlands
- English_Australia
- English_Canada
- English_HongKong
- English_India
- English_Ireland
- English_Jamaica
- English_NewZealand
- English_Singapore
- English_SouthAfrica
- English_UnitedKingdom
- English_UnitedStates
- Estonian_Estonia
- Finnish_Finland
- French_Belgium
- French_Canada
- French_France
- French_Luxembourg
- French_Switzerland
- German_Austria
- German_Germany
- German_Liechtenstein
- German_Luxembourg
- German_Switzerland
- Greek_Greece
- Hebrew_Israel
- Hungarian_Hungary
- Icelandic_Iceland
- Italian_Italy
- Italian_Switzerland
- Japanese_Japan
- Latvian_Latvia
- Lithuanian_Lithuania
- Norwegian_Norway
- Polish_Poland

- Portuguese_Brazil
- Portuguese_Portugal
- Romanian_Romania
- Russian_Russia
- Serbian_Yugoslavia
- Slovak_Slovakia
- Slovenian_Slovenia
- Spanish_Argentina
- Spanish_Bolivia
- Spanish_Chile
- Spanish_Colombia
- Spanish_CostaRica
- Spanish_DominicanRepublic
- Spanish_Ecuador
- Spanish_ElSalvador
- Spanish_Guatemala
- Spanish_Honduras
- Spanish_Mexico
- Spanish_Nicaragua
- Spanish_Panama
- Spanish_Paraguay
- Spanish_Peru
- Spanish_PuertoRico
- Spanish_Spain
- Spanish_UnitedStates
- Spanish_Uruguay
- Spanish_Venezuela
- Swedish_Sweden
- Thai_Thailand
- Turkish_Turkey
- Ukrainian_Ukraine
- Vietnamese_Vietnam

Variable Lists



ENCODING Values

The following tables lists the character encodings that are supported and not supported by WPS.

z/OS encoding values

The following table lists encodings that are supported in the `ENCODING` system option for z/OS servers. They are also available for use with `INFILE` or `FILE` and other language statements for which the `ENCODING` option is valid.

Name	Short Name	Alternative Name	Description
ebcdic037	E037	ibm-037	EBCDIC 037 Old North America
ebcdic273	E273	ibm-273	EBCDIC 273 Old Austria/Germany
ebcdic277	E277	ibm-277	EBCDIC 277 Old Denmark/Norway
ebcdic278	E278	ibm-278	EBCDIC 278 Old Finland/Sweden
ebcdic280	E280	ibm-280	EBCDIC 280 Old Italy
ebcdic284	E284	ibm-284	EBCDIC 284 Old Spain/Latin America
ebcdic285	E285	ibm-285	EBCDIC 285 Old United Kingdom
ebcdic297	E297	ibm-297	EBCDIC 297 Old France
ebcdic420	E420	ibm-420	EBCDIC 420 Old Arabic
ebcdic424	E424	ibm-424	EBCDIC 424 Hebrew
ebcdic500	E500	ibm-500	EBCDIC 500 Old International

Name	Short Name	Alternative Name	Description
ebcdic838	E838	ibm-838	EBCDIC 838 Thai
ebcdic870	E870	ibm-870	EBCDIC 870 Central Europe
ebcdic875	E875	ibm-875	EBCDIC 875 Greek
ebcdic1025	ECYL	ibm-1025	EBCDIC 1025 Cyrillic
ebcdic1026	ETUR	ibm-1026	EBCDIC 1026 Turkish
ebcdic1047	ELAT	ibm-1047	EBCDIC 1047 Latin 1
ebcdic1112	EBAL	ibm-1112	EBCDIC 1112 Baltic
ebcdic1122	EEST	ibm-1122	EBCDIC 1122 Estonian
ebcdic1130	EVIE	ibm-1130	EBCDIC 1130 Vietnamese
ebcdic1140	E140	ibm-1140	EBCDIC 1140 North America
ebcdic1141	E141	ibm-1141	EBCDIC 1141 Austria/Germany
ebcdic1142	E142	ibm-1142	EBCDIC 1142 Denmark/Norway
ebcdic1143	E143	ibm-1143	EBCDIC 1143 Finland/Sweden
ebcdic1144	E144	ibm-1144	EBCDIC 1144 Italy
ebcdic1145	E145	ibm-1145	EBCDIC 1145 Spain/Latin America
ebcdic1146	E146	ibm-1146	EBCDIC 1146 United Kingdom
ebcdic1147	E147	ibm-1147	EBCDIC 1147 France
ebcdic1148	E148	ibm-1148	EBCDIC 1148 International
open_ed-420	EOAR	ibm-420,swaplfnl	Open Edition 420 Old Arabic
open_ed-424	EOIW	ibm-424,swaplfnl	Open Edition 424 Hebrew
open_ed-838	EOTH	ibm-838,swaplfnl	Open Edition 838 Thai
open_ed-875	EOEL	ibm-875,swaplfnl	Open Edition 875 Greek
open_ed-1025	EOCY	ibm-1025,swaplfnl	Open Edition 1025 Cyrillic
open_ed-1026	EOTR	ibm-1026,swaplfnl	Open Edition 1026 Turkish
open_ed-1047	EOL1	ibm-1047,swaplfnl	Open Edition 1047 Latin 1
open_ed-1112	EOBL	ibm-1112,swaplfnl	Open Edition 1112 Baltic
open_ed-1122	EOET	ibm-1122,swaplfnl	Open Edition 1122 Estonian
open_ed-1130	EOVT	ibm-1130,swaplfnl	Open Edition 1130 Vietnamese
open_ed-037	EOUS	ibm-037,swaplfnl	Open Edition 037 Old North America
open_ed-273	EODE	ibm-273,swaplfnl	Open Edition 273 Old Austria/Germany
open_ed-277	EODA	ibm-277,swaplfnl	Open Edition 277 Old Denmark/Norway
open_ed-278	EOFI	ibm-278,swaplfnl	Open Edition 278 Old Finland/Sweden

Name	Short Name	Alternative Name	Description
open_ed-280	EOIT	ibm-280,swaplfnl	Open Edition 280 Old Italy
open_ed-284	EOES	ibm-284,swaplfnl	Open Edition 284 Old Spain
open_ed-285	EOUK	ibm-285,swaplfnl	Open Edition 285 Old United Kingdom
open_ed-297	EOFR	ibm-297,swaplfnl	Open Edition 297 Old France
open_ed-500	EOSW	ibm-500,swaplfnl	Open Edition 500 Old International
open_ed-870	EOL2	ibm-870,swaplfnl	Open Edition 870 Central Europe
open_ed-1140	EO40	ibm-1140,swaplfnl	Open Edition 1140 North America
open_ed-1141	EO41	ibm-1141,swaplfnl	Open Edition 1141 Austria/Germany
open_ed-1142	EO42	ibm-1142,swaplfnl	Open Edition 1142 Denmark/Norway
open_ed-1143	EO43	ibm-1143,swaplfnl	Open Edition 1143 Finland/Sweden
open_ed-1144	EO44	ibm-1144,swaplfnl	Open Edition 1144 Italy
open_ed-1145	EO45	ibm-1145,swaplfnl	Open Edition 1145 Spain
open_ed-1146	EO46	ibm-1146,swaplfnl	Open Edition 1146 United Kingdom
open_ed-1147	EO47	ibm-1147,swaplfnl	Open Edition 1147 France
open_ed-1148	EO48	ibm-1148,swaplfnl	Open Edition 1148 International
ibm-930	J930	ibm-930	Katakana (IBM)
ibm-933	KIBM	ibm-933	Korean (IBM)
ibm-935	ZIBM	ibm-935	Simplified Chinese (IBM)
ibm-937	YIBM	ibm-937	Traditional Chinese (IBM)
ibm-939	JIBM	ibm-939	Japanese (IBM)

Encoding values on non-z/OS platforms

The following table lists encodings that are supported in the `ENCODING` system option for other platform servers, such as Windows and Linux. They are also available for use with `INFILE` or `FILE` and other language statements for which the `ENCODING` option is valid.

Name	Short Name	Alternative Name	Description
utf-8	UTF8	utf-8	UTF-8 Unicode
ibm-942	JPC	ibm-942	Japanese (PCIBM)
ibm-949	KPC	ibm-949	Korean (PCIBM)
ibm-950	ZPC	ibm-950	Traditional Chinese (PCIBM)

Name	Short Name	Alternative Name	Description
us-ascii	ANSI	us-ascii	7-Bit US-ASCII
latin1	LAT1	iso-8859-1	ISO-8859-1 Western
latin2	LAT2	iso-8859-2	ISO-8859-2 Central Europe
latin3	LAT3	iso-8859-3	ISO-8859-3 Southern Europe
latin4	LAT4	iso-8859-4	ISO-8859-4 Northern Europe
latin5	LAT5	iso-8859-9	ISO-8859-9 Turkish
latin6	LAT6	iso-8859-10	ISO-8859-10 Baltic
latin9	LAT9	iso-8859-15	ISO-8859-15 European
cyrillic	CYRL	iso-8859-5	ISO-8859-5 Cyrillic
arabic	ARAB	iso-8859-6	ISO-8859-6 Arabic
greek	GREK	iso-8859-7	ISO-8859-7 Greek
hebrew	HEBR	iso-8859-8	ISO-8859-8 Hebrew
thai	THAI	iso-8859-11	ISO-8859-11 Thai
pcoem437	P437	ibm-437	IBM-437 USA
pcoem850	P850	ibm-850	IBM-850 Western
pcoem852	P852	ibm-852	IBM-852 Central Europe
pcoem857	P857	ibm-857	IBM-857 Turkish
pcoem858	P858	ibm-858	IBM-858 European
pcoem862	P862	ibm-862	IBM-862 Hebrew
pcoem864	P864	ibm-864	IBM-864 Arabic
pcoem865	P865	ibm-865	IBM-865 Nordic
pcoem866	P866	ibm-866	IBM-866 Cyrillic
pcoem869	P869	ibm-869	IBM-869 Greek
pcoem874	P874	ibm-874	IBM-874 Thai
pcoem921	P921	ibm-921	IBM-921 Baltic
pcoem922	P922	ibm-922	IBM-922 Estonian
pcoem1129	PVIE	ibm-1129	IBM-1129 Vietnamese
msdos720	P720	windows-720	Windows OEM 720 Arabic
msdos737	P737	windows-737	Windows OEM 737 Greek
msdos775	P775	windows-775	Windows OEM 775 Baltic
pcoem860	P860	ibm-860	IBM-860 Portuguese (MS-DOS)
pcoem863	P863	ibm-863	IBM-863 French Canadian

Name	Short Name	Alternative Name	Description
wlatin2	WLT2	windows-1250	Windows-1250 Central Europe
wcyrillic	WCYL	windows-1251	Windows-1251 Cyrillic
wlatin1	WLT1	windows-1252	Windows-1252 Western
wgreek	WBRK	windows-1253	Windows-1253 Greek
wturkish	WTUR	windows-1254	Windows-1254 Turkish
whebrew	WHEB	windows-1255	Windows-1255 Hebrew
warabic	WARA	windows-1256	Windows-1256 Arabic
wbaltic	WBAL	windows-1257	Windows-1257 Baltic
wvietnamese	WVIW	windows-1258	Windows-1258 Vietnamese
ms-950	ZWIN	windows-950	Traditional Chinese (PCMS)
euc-cn	ZEUC	euc-cn	Simplified Chinese (EUC)
euc-tw	YEUC	euc-tw	Traditional Chinese (EUC)
big5	BIG5	big5	Traditional Chinese (Big5)
ms-936	YWIN	windows-936	Simplified Chinese (PCMS)
euc-jp	JEUC	euc-jp	Japanese (EUC)
ms-932	J932	windows-932	Japanese (PCMS)
shift-jis	SJIS	shift_jis	Japanese (SJIS)
euc-kr	KEUC	euc-kr	Korean (EUC)
ms-949	KWIN	windows-949	Korean (PCMS)

Not supported for server encoding

The following table lists encodings that are not supported for server encoding, but can be used for the import and export of data.

Name	Short Name	Alternative Name	Description
utf-16be		utf-16BE	UTF-16 Big Endian Unicode
utf-16le		utf-16LE	UTF-16 Little Endian Unicode
utf-32be		utf-32BE	UTF-32 Big Endian Unicode
utf-32le		utf-32LE	UTF-32 Little Endian Unicode

LOCALE Values

The following table lists supported LOCALE values, and PXLOCALE and PXREGION codes. The PXLOCALE code is created from a combination of a language code for the locale and the locale region code.

LOCALE values	PXLOCALE	PXREGION
Afrikaans_SouthAfrica	af_ZA	ZA
Albanian_Albania	sq_AL	AL
Arabic_Algeria	ar_DZ	DZ
Arabic_Bahrain	ar_BH	BH
Arabic_Egypt	ar_EG	EG
Arabic_India	ar_IN	IN
Arabic_Iraq	ar_IQ	IQ
Arabic_Jordan	ar_JO	JO
Arabic_Kuwait	ar_KW	KW
Arabic_Lebanon	ar_LB	LB
Arabic_Libya	ar_LY	LY
Arabic_Morocco	ar_MA	MA
Arabic_Oman	ar_OM	OM
Arabic_Qatar	ar_QA	QA
Arabic_SaudiArabia	ar_SA	SA
Arabic_Sudan	ar_SD	SD
Arabic_Syria	ar_SY	SY
Arabic_Tunisia	ar_TN	TN
Arabic_UnitedArabEmirates	ar_AE	AE
Bengali_India	bn_IN	bn
Bosnian_BosniaHerzegovina	bs_BA	BA
Bulgarian_Bulgaria	bg_BG	BG
Byelorussian_Belarus	be_BY	BY
Catalan_Spain	ca_ES	ES
Chinese_China	zh_CN	CN
Chinese_HongKong	zh_HK	HK
Chinese_Macau	zh_MO	MO
Chinese_Singapore	zh_SG	SG

LOCALE values	PXLOCALE	PXREGION
Chinese_Taiwan	zh_TW	TW
Catalan_Spain	ca_ES	ES
Cornish_UnitedKingdom	kw_GB	GB
Croatian_BosniaHerzegovina	hr_BA	BA
Croatian_Croatia	hr_HR	HR
Czech_CzechRepublic	cs_CZ	CZ
Danish_Denmark	da_DK	DK
Dutch_Belgium	nl_BE	BE
Dutch_Netherlands	nl_NL	NL
English_Australia	en_AU	AU
English_Belgium	en_BE	BE
English_Botswana	en_BW	BW
English_Canada	en_CA	CA
English_Caribbean	en_CB	CB
English_HongKong	en_HK	HK
English_India	en_IN	IN
English_Ireland	en_IE	IE
English_Jamaica	en_JM	JM
English_Malta	en_MT	MT
English_NewZealand	en_NZ	NZ
English_Philippines	en_PH	PH
English_Singapore	en_SG	SG
English_SouthAfrica	en_ZA	ZA
English_UnitedKingdom	en_GB	GB
English_UnitedStates	en_US	US
English_Zimbabwe	en_ZW	ZW
Estonian_Estonia	et_EE	EE
Faroese_Faroelands	fo_FO	FO
Finnish_Finland	fi_FI	FI
French_Belgium	fr_BE	BE
French_Canada	fr_CA	CA
French_France	fr_FR	FR
French_Luxembourg	fr_LU	LU

LOCALE values	PXLOCALE	PXREGION
French_Switzerland	fr_CH	CH
German_Austria	de_AT	AT
German_Germany	de_DE	DE
German_Liechtenstein	de_LI	LI
German_Luxembourg	de_LU	LU
German_Switzerland	de_CH	CH
Greek_Cyprus	el_CY	CY
Greek_Greece	el_GR	GR
Greenlandic_Greenland	kl_GL	GL
Hebrew_Israel	he_IL	IL
Hindi_India	hi_IN	IN
Hungarian_Hungary	hu_HU	HU
Icelandic_Iceland	is_IS	IS
Indonesian_Indonesia	id_ID	ID
Irish_Ireland	ga_IE	IE
Italian_Italy	it_IT	IT
Italian_Switzerland	it_CH	CH
Japanese_Japan	ja_JP	JP
Korean_Korea (South Korea)	ko_KR	KR
Latvian_Latvia	lv_LV	LV
Lithuanian_Lithuania	lt_LT	LT
Macedonian_Macedonia	mk_MK	MK
Malay_Malaysia	ms_MY	MY
Maltese_Malta	mt_MT	MT
ManxGaelic_UnitedKingdom	gv_GB	GB
Marathi_India	mr_IN	IN
NorwegianBokmal_Norway	nb_NO	NO
NorwegianNynorsk_Norway	nn_NO	NO
Norwegian_Norway	no_NO	NO
Persian_India	fa_IN	IN
Persian_Iran	fa_IR	IR
Polish_Poland	pl_PL	PL
Portuguese_Brazil	pt_BR	BR

LOCALE values	PXLOCALE	PXREGION
Portuguese_Portugal	pt_PT	PT
Romanian_Romania	ro_RO	RO
Russian_Russia	ru_RU	RU
Russian_Ukraine	ru_UA	UA
Serbian_BosniaHerzegovina	sr_BA	BA
Serbian_Montenegro	sr_ME	ME
Serbian_Serbia	sr_RS	RS
SerbianLatin_Serbia	sh_RS	RS
SerbianLatin_BosniaHerzegovina	sh_BA	BA
SerbianLatin_Montenegro	sh_ME	ME
Slovak_Slovakia	sk_SK	SK
Slovenian_Slovenia	sl_SI	SI
Spanish_Argentina	es_AR	AR
Spanish_Bolivia	es_BO	BO
Spanish_Chile	es_CL	CL
Spanish_Colombia	es_CO	CO
Spanish_CostaRica	es_CR	CR
Spanish_DominicanRepublic	es_DO	DO
Spanish_Ecuador	es_EC	EC
Spanish_ElSalvador	es_SV	SV
Spanish_Guatemala	es_GT	GT
Spanish_Honduras	es_HN	HN
Spanish_Mexico	es_MX	MX
Spanish_Nicaragua	es_NI	NI
Spanish_Panama	es_PA	PA
Spanish_Paraguay	es_PY	PY
Spanish_Peru	es_PE	PE
Spanish_PuertoRico	es_PR	PR
Spanish_Spain	es_ES	ES
Spanish_UnitedStates	es_US	US
Spanish_Uruguay	es_UY	UY
Spanish_Venezuela	es_VE	VE

LOCALE values	PXLOCALE	PXREGION
Swedish_Sweden	sv_SE	SE
Tamil_India	ta_IN	IN
Telugu_India	te_IN	IN
Thai_Thailand	th_TH	TH
Turkish_Turkey	tr_TR	TR
Ukrainian_Ukraine	uk_UA	UA
Vietnamese_Vietnam	vi_VN	VN

WPS Core

System options

System options are settings that apply to a WPS Analytics.

System options enable you to set the behaviour of WPS and the language of SAS. For example, you can set system options that affect how errors are handled, how datasets and external files are handled, how sorts are performed, and so on.

System options can be set in a SAS-language program, in a SAS-language configuration file, or on the command line.

The reference documentation for each system option:

- Assumes that system options can be set on all operating systems, unless specified otherwise in a *Supported Items* section.
- Specifies whether it is:
 - Portable – The system option can be moved to a different operating system to the one it was set on.
 - Restrictable – The system option can be restricted. See [Restricting system options](#) (page 46) for more details.
 - Saveable – The system option can be saved. See [OPTSAVE](#) (page 2408) for more details.

Setting system options

You can set system options in various ways: in a configuration file, on the WPS command line, and by using the `OPTIONS` statement in a Sas-language program.

You can set system options using:

- A configuration file. Configuration files are read when Workbench starts, or when WPS is run from the command line, or if specified on the command line. For information on configuration files, see *Configuration files* in the **WPS Analytics installation and administration guide**.
- The command line. The system option remains in effect while the program runs. For information on running programs on the command line and setting system options, see *Command line mode* in *Run a program* in the **WPS Workbench User Guide**.
- The `OPTIONS` (page 619) global statement.

Once set, a system option remains in effect for the session. The value of a system option is the last value set. If you set a system option in a configuration file, but subsequently set it on the command line, the value of the system option on the command line is used. If you set a system option on a command line, but the system option is set in a SAS language program, then the value in the program is used.

Some system options can only be set on the command line or in a configuration file because they affect the environment of the WPS session. For example, the system option `ENCODING`, which specifies the character encoding of the WPS system, can only be set on the command line or in a configuration file because the encoding for the session needs to be specified before a program is run. The description for each system option specifies how it can be set.

Specifying configuration files

A configuration file can be used by specifying it on the command line, or in another configuration file.

There is a default configuration file that is included with the WPS installation, called `wps.cfg`, located in the WPS installation directory. We recommend that you do not edit this file. Instead, create a configuration file that itself opens `wps.cfg` before any other system options are set. Because system options are set in the order in which they are specified, all default options in `wps.cfg` are set first, and then the system options in the configuration file that calls `wps.cfg`. A configuration file can contain references configuration files other than `wps.cfg`, and can contain as many references as required.

Configuration files are specified with the option:

```
-CONFIG filename
```

where *filename* is the filename of the configuration file, including an absolute or relative path.

In the following example, the option is specified on the command line. The configuration file `ycopt.cfg` contains:

```
-config 'C:\Program Files\World Programming\WPS\4\wps.cfg'  
-yearcutoff 1900
```

The first entry in the file is the option `CONFIG`, specifying the default configuration file; this ensures that all default system options are set. The `YEARCUTOFF` option is then set.

The following program (`c:\temp\test.wps`) is then run on the command line:

```
DATA _NULL_;  
  x = '16-JUN-02'D;  
  PUT x DATE11.;  
RUN;
```

The `CONFIG` option is also set on the command line to specify the file `ycopt.cfg`:

```
WPS c:\temp\test.wps -CONFIG c:\temp\ycopt.cfg
```

The following result is written to the log:

```
16-JUN-1902
```

The specified configuration file has been used to set the system option. The last system option read in the configuration file `ycopt.fcgi` is `YEARCUTOFF 1900`, which sets the starting year for interpreting two digit years as 1900. The date `16-JUN-02` is therefore interpreted as `16-JUN-1902`.

Setting system options on the command line

On the command line, an option specified to a system option is treated as a single string. An option might, therefore, need to be escaped or entered in quotation marks (depending on the operating system) if it contains special characters, including spaces.

Some system options enable you to specify more than one option, in which case the options are separated by spaces and enclosed in brackets. For example, the system option `AUTOEXEC` enables you to specify programs that run when WPS starts. You can specify more than one program to start, if required:

```
WPS test.wps -AUTOEXEC "(test1.wps test2.wps)"
```

WPS first runs `test1.wps` and `test2.wps` before running `test.wps`. However, for this command to execute on Windows, the option must be enclosed in quotation marks, as the option is treated as one string. Without the quotation marks, Windows assumes that the option finished at the space, and `test2.wps` would be an unrecognised option on the command line.

Setting system options in Workbench

WPS Workbench also provides dialogues in which system options can be set. For information, see *WPS server properties* in *WPS Server Explorer* in *WPS server* in your **WPS Workbench User Guide**.

Appending and inserting values

You can modify many system options that take string values using the `APPEND` and `INSERT` directives. These directives can be specified as arguments to the `OPTION` statement, as arguments on the WPS command line, or in a configuration file. They have the same format as system options.

`APPEND` and `INSERT` can be used with any system option that has the attribute `Appendable`.

`APPEND` enables you to add a string to the end of the current value of a system option. `INSERT` enables you to add a string at the beginning of the current value of a system option. These options have the format.

For more information on the format of `APPEND` and `INSERT`, see the `OPTIONS` [🔗](#) (page 619) statement.

For example, to append a value in the following program (`myprog.wps`):

```
LIBNAME MS1 "c:\wk\macros1";  
OPTIONS APPEND = (SASAUTOS = MS1);  
%mymac;
```

If `SASAUTOS` is already specified in a configuration file with the value `"c:\wk\macros2"`, the new value when the program is run is:

```
SASAUTOS = ("c:\wk\macros2" ms1)
```

You could subsequently change that value of `SASAUTOS` on the command line, for example, so that another value is added when the program is run, in this example on a Windows command line:

```
wps myprog -insert '(sasautos = ("c:\temp\macros3" "c:\temp\wkmacros"))'
```

The new value of `SASAUTOS` is then:

```
SASAUTOS = ("c:\temp\macros3" "c:\temp\wkmacros" "c:\wk\macros2" MS1)
```

where:

- "c:\temp\macros3" and "c:\temp\wkmacros" have been inserted at the beginning of the system option from the command line.
- "c:\wk\macros2" was specified in a configuration file.
- The library name `MS1` has been appended by the `OPTIONS` statement in the program.

Displaying system options

You might want to view which system options have been set. You can use the `OPTIONS` procedure to do this.

`PROC OPTIONS` [↗](#) (page 619) can be used to list the current settings of all options, or the setting of a specific option. You can specify the amount and kind of detail that is returned.

To list the current settings of all system options, and return summary information about what they do, enter:

```
PROC OPTIONS;
```

The procedure lists the options in two groups, *host* and *portable* options. Host options only have an effect on the current system, while portable options have an effect on the current system and on any operating system to which the SAS language program is ported.

For example, when you run the `OPTION` procedure the following might be listed in the log. Each section of the listing has been truncated for brevity.

Portable Options:

```
ALTLOG=           Name of the altlog
NOAUTOSIGNON      Remote submit will not attempt to automatically signon
BASEENGINE=WPD    The library engine to use when BASE is specified
BOMFILE           Write a Byte Order Mark prefix on external Unicode files
BUFNO=1           Specifies the number of buffers used by a library engine for
a data set (not honoured by all engines)
BUFSIZE=0         Specifies the size of a page for a WPS data set
:
```

Host Options:

```
AUTOEXEC=         The location of a file automatically executed at WPS
initialisation
BOTTOMMARGIN=1CM  The bottom margin of a page output using the ods pdf
destination
```

```
CPUCOUNT=8      Number of CPUs available to the application
NOBIDIRECTEXEC   PROC SQL will handle CREATE TABLE AS SELECT and DELETE
statements to a relational database in the normal way
NODLCREATEDIR    Do not allow a directory to be created upon assignment of a
libname, even if the directory does not already exist
NODMS           Use the non-windowed environment
:
```

Restricting system options

You can restrict a system option so that it cannot be changed. This enables specific values to be enforced by the system administrator.

System options can be restricted by using the:

- **RESTRICT** option of the **OPTIONS** global statement
- **RESTRICT** option in configuration files
- Restricted configuration file `rwps.cfg` (Windows, UNIX and Linux only)
- **RESTRICT** option of the **SITEOPTION** statement of **PROC SETINIT**

Using the **RESTRICT** option of the **OPTIONS** statement only restricts specified system options during the session in which the statement is set, and is more appropriate for users. To restrict system options more generally, use one of the methods described below.

Not all system options can be restricted. Whether a system option can be restricted is indicated in the description for each system option.

Using the **RESTRICT** configuration option

The **RESTRICT** keyword can be used in configuration files. System options can be restricted using the format:

```
RESTRICT = sys-options
```

on z/OS, or:

```
-RESTRICT sys-options
```

on other operating systems. *sys-options* can be a single system option, or a list of space-separated system options enclosed in parentheses. For example:

```
-RESTRICT MEMSIZE
```

or:

```
-RESTRICT (MEMSIZE CARDIMAGE NODATE)
```

Using a restricted configuration file

On Windows, UNIX and Linux, you can restrict system options by entering them in the restricted configuration file. The file is read when the WPS session starts. System options set in the restricted configuration file cannot be subsequently changed by users during the session.

The file has the name:

```
wps\home\misc\rstropts\rwps.cfg
```

where *wps\home* is the installation folder or directory of WPS. For example, the default installation folder for Windows is:

```
C:\Program Files\World Programming\WPS\4
```

The path and filename of the restricted file is therefore:

```
C:\Program Files\World Programming\WPS\4\misc\rstropts\rwps.cfg
```

Note:

System options cannot be restricted in this way on z/OS when WPS is run as an MVS job or by TSO in the foreground.

On UNIX and Linux you can also restrict the configuration file to particular groups or to particular users:

```
wps\home\misc\rstropts\name_rwps.cfg
```

where *name* is group or user name.

Using the SETINIT procedure

You cannot restrict the ability to change system options by using a restricted configuration file on z/OS. If you want to ensure that users cannot modify a system option on z/OS you must use the SETINIT procedure.

The SETINIT procedure is defined in the WPS license text file. When WPS starts, it reads the installation file generated when the SETINIT procedure runs. The specified system options are then set for all WPS installations to which the license applies.

System options are restricted using the RESTRICT option of the SITEOPTION statement of PROC SETINIT.

For example, a site administrator could ensure users never see the logic of macros that are being used by adding the following lines to the license file before the license is applied to a WPS installation:

```
PROC SETINIT;
... statements in SETINIT...
SITEOPTION = "NOMLOGIC RESTRICT MLOGIC;";
SAVE;
RUN;
```

When the license is applied PROC SETINIT is run; the NOMLOGIC system option is set, and then MLOGIC restricted. Users can then never change the setting using, for example, the OPTIONS statement.

Finding out if an option is restricted

You can find out if an option is currently restricted using the `GETOPTION` function. This has an optional argument that you can set to `HOWRESTRICTED` or `HOWSET`. If set to `HOWRESTRICTED`, a string is returned indicating whether the option is restricted. For example:

```
DATA _NULL_;
  isr = getoption("SPOOL", "HOWRESTRICTED");
  put isr;
RUN;
```

The following is written to the log:

```
Unrestricted
```

For information on the options, see [GETOPTION](#) (page 2114).

COMMUNICATIONS group system options

The system options in this group specify settings for WPS Communicate. Some of these options specify values that provides default values for WPS Communicate statement options. For example, you can specify the remote server to which a Communicate connection is made using either the `CONNECTREMOTE` system option, or the `CONNECTREMOTE` option of the `RSUBMIT` statement.

AUTOSIGNON

Specifies whether remote submit (`RSUBMIT`) automatically attempts to sign on.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOAUTOSIGNON
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

AUTOSIGNON

Remote submit automatically attempts to sign on.

NOAUTOSIGNON

Remote submit does not automatically attempt to sign on.

If you do not specify this option, you need two WPS Communicate statements to initiate a remote submit, `SIGNON` and `RSUBMIT`. If you specify this option, you need only specify `RSUBMIT`.

If you specify `AUTOSIGNON` you must also specify the `CONNECTREMOTE` [↗](#) (page 52) system option to identify the server to which the remote submit will connect.

Other sign-on options that are set with the `SIGNON` statement can also be set with system options in the `COMMUNICATIONS` group. Other options of `SIGNON`, such as authentication, can be set in `RSUBMIT`.

The `SIGNON` and `RSUBMIT` statements are described in the **WPS Communicate User Guide and Reference**.

Example

In this example, the `OPTIONS` statement is used to specify that an automatic sign-on is attempted when `RSUBMIT` is encountered.

```
OPTIONS AUTOSIGNON CONNECTREMOTE=myremote;
RSUBMIT USERNAME="jdoe" PASSWORD="xxxxxxxx"
  LAUNCHCMD=' /opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

COMAMID

Specifies the communication method to use for establishing remote communications.

⇒ **COMAMID** ⇒ = ⇒ *communication-method* ⇒

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>TCP</code>
Maximum length:	8
Option group:	<code>COMMUNICATIONS</code>
Portable	True
Restrictable	False
Saveable	False

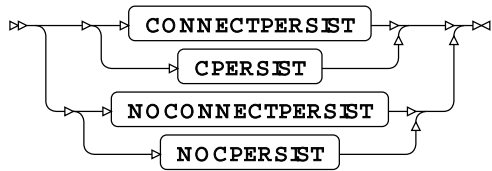
communication-method

The supported communication method; currently only `TCP` is supported.

- `TCP`

CONNECTPERSIST

Specifies whether a remote connection persists after an `ENDRSUBMIT` statement.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOCONNECTPERSIST</code>
Option group:	<code>COMMUNICATIONS</code>
Portable	True
Restrictable	True
Saveable	True

CONNECTPERSIST

The remote connection persists.

NOCONNECTPERSIST

The remote connection does not persist.

The connection created for an `RSUBMIT` block normally ends when `ENDRSUBMIT` is executed. If `CONNECTPERSIST` is specified, the connection persists, and work folders and resources remain available, and can be used by subsequent `RSUBMIT` blocks.

Note:

The connection will not persist after a `SIGNOFF` statement, even if `CONNECTPERSIST` is specified.

The setting of this option can be overridden in a program using the `PERSIST` option of the `RSUBMIT` statement.

`RSUBMIT` blocks, and the `RSUBMIT` and `ENDRSUBMIT` statements, are described in the **WPS Communicate User Guide and Reference**.

Basic example

In this example, the `OPTIONS` statement is used to specify that connections do not continue to exist after `ENDRSUBMIT` statements.

```
OPTIONS NOCONNECTPERSIST;
SIGNON
  CONNECTREMOTE = myserver
  USERNAME = "jdoe"
  PASSWORD = "xxxxxxxx"
  LAUNCHCMD='/opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, again";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

The second `RSUBMIT` block is not executed because the `NOCONNECTPERSIST` system option is set. Error messages are displayed in the log. If `CONNECTPERSIST` had been set, both blocks would run.

Example – autosignon and connection persistence

In this example, the `OPTIONS` statement is used to specify that connections do continue to exist after `ENDRSUBMIT` statements. The connection is made through the `RSUBMIT` statement after the `AUTOSIGNON` system option has been specified.

```
OPTIONS CONNECTPERSIST AUTOSIGNON;
RSUBMIT
  CONNECTREMOTE = lcen7x64d01 USERNAME = "jlees"
  PASSWORD = "xXy689001k"
  LAUNCHCMD='/opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;

RSUBMIT;
  DATA _NULL_;
  PUT "Hello, again";
  RUN;
ENDRSUBMIT;

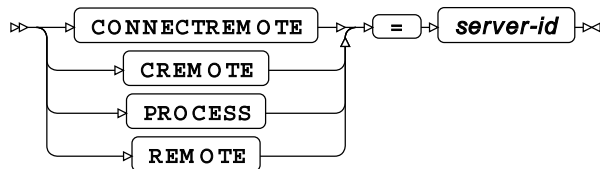
RSUBMIT;
  DATA _NULL_;
  PUT "One more time";
  RUN;
ENDRSUBMIT;

SIGNOFF;
```

The second and third `RSubmit` blocks are executed because the `CONNECTPERSIST` system option is set. They use the sign-on information of the first `RSubmit`.

CONNECTREMOTE

Specifies the remote server to which a connection is made in WPS Communicate.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	1024
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

server-id

The name or IP address of the remote server.

If this system option is set, you do not have to specify a connection in a `SIGNON` or `RSubmit` statement to sign on to a remote server. If you do specify a connection in one of those statements, it will override the connection specified by this system option.

The `RSubmit` and `SIGNON` statements, are described in the ***WPS Communicate User Guide and Reference***.

Example

In this example, the `OPTIONS` statement is used to specify the name of the server for a WPS Communicate session.

```
OPTIONS CONNECTREMOTE=myremote;
SIGNON
  USERNAME = "jdoe"
  PASSWORD = "xxxxxxxxx"
  LAUNCHCMD='/opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

Because the connection is specified using the system option, it does not need to be specified in the `SIGNON` statement.

CONNECTTIMEOUT

Specifies a WPS Communicate client time-out for connect and read operations.

➤ **CONNECTTIMEOUT** ➤ = ➤ *timeout* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	60000
Minimum value:	0
Maximum value:	3600000
Option group:	COMMUNICATIONS
Portable	True

If a response from the remote server is not received within the specified timeout period, an error is written to the log and the attempt to connect terminates.

timeout

The timeout, in milliseconds.

If you specify 0 (zero), no timeout is set.

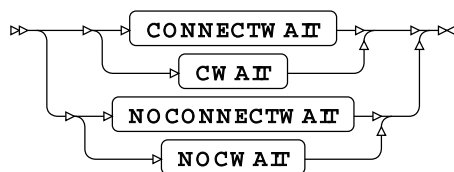
Example

In this example, the `OPTIONS` statement is used to specify that an automatic sign-on is attempted when `RSUBMIT` is encountered, but that if no response is received in 30 seconds, the program terminates.

```
OPTIONS CONNECTTIMEOUT=30000 AUTOSIGNON CONNECTREMOTE=myremote;
RSUBMIT USERNAME="jdoe" PASSWORD="xxxxxxxx"
  LAUNCHCMD=' /opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

CONNECTWAIT

Specifies whether `RSUBMIT` blocks are executed synchronously or asynchronously.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOCONNECTWAIT</code>
Option group:	<code>COMMUNICATIONS</code>
Portable	True
Restrictable	True
Saveable	True

CONNECTWAIT

Submit `RSUBMIT` blocks synchronously; blocks will not wait for preceding blocks to finish executing.

NOCONNECTWAIT

Submit `RSUBMIT` blocks asynchronously; a block only starts when the preceding block has finished executing.

`RSUBMIT` blocks can be run asynchronously, that is, at the same time, or synchronously, that is one after another. By default If `NOCONNECTWAIT` is specified, WPS will not wait for one `RSUBMIT` block to end before starting another.

The system option can be overridden by the `WAITFOR` statement in an `RSUBMIT` block.

Example

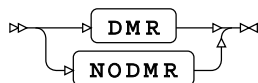
In this example, the `OPTIONS` statement is used to specify that `RSUBMIT` blocks can run asynchronously.

```
OPTIONS NOCONNECTWAIT;  
SIGNON  
  CONNECTREMOTE = myremote  
  USERNAME = "zx"  
  PASSWORD = "*****"  
  LAUNCHCMD='/opt/worldprogramming/wps-4/bin/wps -dmr '  
  SSH;  
RSUBMIT;  
  DATA _NULL_;  
  do i = 1 to 1000000000;  
  end;  
  PUT "Hello, from the remote computer";  
  RUN;  
ENDRSUBMIT;  
RSUBMIT;  
  DATA _NULL_;  
  PUT "Hello, again";  
  RUN;  
ENDRSUBMIT;  
SIGNOFF;
```

The second block will begin to execute before the first block has finished.

DMR

Specifies whether to invoke a remote or local WPS Communicate session.



Valid in:	Command line.
Default:	NODMR
Option group:	COMMUNICATIONS
Portable	True
Restrictable	False
Saveable	False

DMR

Starts a WPS session on a remote service using Communicate. A WPS client can then run a program using WPS on the server.

NODMR

A WPS session is not established, and an error message is written to the log.

This system option must only be used on the command line by specifying it to the:

- SASCMD or LAUNCHCMD option of the SIGNON or RSUBMIT statement.
- SASCMD system option.

To use WPS on a remote server through Communicate, this option must be specified. This starts the remote session in server mode and returns the session connection details to the initiating WPS client. If it is not specified, or if NODMR is specified, a connection is not established, and an error message is written to the log.

For example, to start the WPS Communicate and use the `wps.exe` command in `/opt/worldprogramming/wps-4/bin/`:

```
SASCMD = '/opt/worldprogramming/wps-4/bin/wps -DMR'
```

Note:

If you specify `-DMR` on the operating system command line to start a program, a communication session is established to an assigned port, but you cannot do anything with the session, and the program will not run. For example, if you specify:

```
wps test2.wps -dmr
```

a WPS Communicate session starts:

```
WPS COMMUNICATE PORT=54496      SESSION ESTABLISHED.
```

However, the program `test2` does not run. You can do nothing with the session and you will have to stop the process.

Example

In this example, the `DMR` option is specified to the `LAUNCHCMD` option of the `SIGNON` statement.

```
OPTIONS SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr';
SIGNON
  CONNECTREMOTE = myserver
  USERNAME = "jdoe"
  PASSWORD = "xxxxxxxxxx"
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "I'm running over here";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

You can then run this program from the command line; for example:

```
wps test2.wps
```

SASCMD

Specifies the command required to start WPS on the remote system when Communicate signs on.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	32767
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

command

The command to use. For example, on a Linux system this might be:

```
' /opt/worldprogramming/wps-4/bin/wps '
```

On a Windows system , this might be:

```
'C:\Program Files\World Programming\WPS\4\bin\wps '
```

The setting of this option can be overridden in a program by the SASCMD option of the RSUBMIT or SIGNON statement.

For information on RSUBMIT blocks, and how you specify remote servers, see the **WPS Communicate User Guide and Reference**.

Note:

You must always also specify the DMR system option on the command line used to start the remote server session in Communicate; see DMR [🔗](#) (page 55) for information.

Example

In this example, the `OPTIONS` statement is used to specify the location of the WPS executable command.

```
OPTIONS SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr';
SIGNON
  CONNECTREMOTE = myremote
  USERNAME = "jdoe"
  PASSWORD = "x2x3x4x1jd"
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

SASSCRIPT

Specifies the location of WPS Communicate telnet sign-on scripts.

➤ **SASSCRIPT** ➤ = ➤ *filepath* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	1024
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

filepath

The path to the telnet script location.

This system option can be used to specify the location of a telnet script specified by the `CSCRIPT` option of the `RSUBMIT` or `SIGNON` statements. This script is used to control the login process for telnet session.

If you specify the filepath to a script using the `CSCRIPT` option of the `RSUBMIT` or `SIGNON` statements, this system option is ignored.

For information on `RSUBMIT` and `SIGNON`, see the ***WPS Communicate User Guide and Reference***.

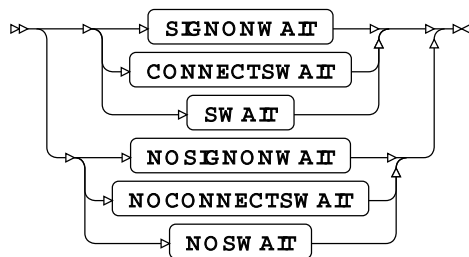
Example

In this example, the `OPTIONS` statement is used to specify the `SASSCRIPT` option. The telnet login script `tnlogin` is found in the folder `c:\scripts`.

```
OPTIONS SASSCRIPT=c:\scripts;
SIGNON
  CONNECTREMOTE = myserver
  USERNAME = "jdoe"
  PASSWORD = "x1x2x3!!"
  SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr'
  cscript=tnlogin
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

SIGNONWAIT

Specifies whether sign-on is complete before executing subsequent program statements.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOSIGNONWAIT`

Option group: `COMMUNICATIONS`

Portable: `True`

Restrictable: `True`

Saveable: `True`

SIGNONWAIT

Wait for sign-on to complete.

NOSIGNONWAIT

Do not wait for sign-on to complete.

If `SIGNONWAIT` is specified, no program statements are executed until the WPS Communicate sign-on process has finished. If `NOSIGNONWAIT` is specified, statements subsequent to any sign-on statements can be executed before the WPS Communicate sign-on process has finished.

The system option can be overridden by the `SIGNONWAIT` option of the `RSUBMIT` or `SIGNON` statements.

Example

In this example, the `OPTIONS` statement is used to specify that subsequent program statements will not wait for sign-on to complete.

```
OPTIONS SSH_HOSTVALIDATION = NONE NOSIGNONWAIT;
SIGNON
  CONNECTREMOTE = myserver
  USERNAME = 'jdoe'
  PASSWORD = 'x1x2x3x1!'
  SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr'
  SSH;
DATA _NULL_;
  DO i = 1 TO 1000000;
    x = (i + 1)/20;
  END;
  PUT 'End of this program';
RUN;
RSUBMIT;
  DATA _NULL_;
    PUT 'Hello, from the remote computer';
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

The statements in the `DATA` step after the `SIGNON` statement will not wait for the sign-on process to complete before executing.

SSH_HOSTVALIDATION

Specifies the method used to validate an SSH host key.

➤ `SSH_HOSTVALIDATION` ➤ `=` ➤ *validation-method* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	16
Option group:	<code>COMMUNICATIONS</code>
Portable	True
Restrictable	True
Saveable	True

When an SSH connection is made to a remote server, the identity of the remote server is validated using either the OpenSSH or the PuTTY validation methods, by searching for the server in either the Open SSH or the PuTTY known hosts files. If the host is not known, then the server connection will not proceed unless the `NONE` option is specified.

validation-method

The type of validation:

- `NONE`

No validation is performed. Only use this option in a test environment.

- `OPENSSH`.

SSH is validated using OpenSSH. This is the default method for all operating systems except Windows. All remote servers must be known to OpenSSH.

- `PUTTY`.

SSH is validated using PuTTY. This is the default method for Windows. All remote servers must be known to PuTTY.

Basic example

In this example, the `OPTIONS` statement is used to specify that no validation is used.

```
OPTIONS SSH_HOSTVALIDATION = NONE;
SIGNON
  CONNECTREMOTE = myremote
  USERNAME = "jdoe"
  PASSWORD = "x1x2x3x4"
  SASCMD=' /opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

Example - server not in PuTTY host file

In this example, the `OPTIONS` statement is used to specify that the PuTTY host file is used, but the server does not the server.

```
OPTIONS SSH_HOSTVALIDATION = PuTTY;
SIGNON
  CONNECTREMOTE = myremote
  USERNAME = "jdoe"
  PASSWORD = "x1x2x3x4"
  SASCMD=' /opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
RSUBMIT;
  DATA _NULL_;
  PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

This produces the following output:

```
NOTE: Remote SSH signon to myremote starting
ERROR: SSH error occurred : reject HostKey: LCEN7X64D01
NOTE: Remote signon to myremote failed
```

SSH_IDENTITYFILE

Specifies the location of the SSH identity file used for public key authentication.

⇒ **SSH_IDENTITYFILE** ⇒ = ⇒ *filepath* ⇒

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	32767
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

filepath

The filename and pathname of the identity file.

The system option can be overridden by the `IDENTITYFILE` option of the `RSUBMIT` or `SIGNON` statements.

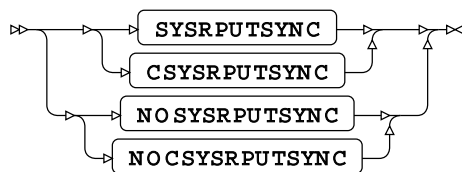
Example

In this example, the `OPTIONS` statement is used to specify the filename of an SSH identity file.

```
OPTIONS SSH_IDENTITYFILE = 'c:\ssh\ssh-files\id-file';
SIGNON
  CONNECTREMOTE = myremote
  USERNAME = "fdoe"
  PASSWORD = "xyxyxyxyx"
  SASCMD=' /opt/worldprogramming/wps-4/bin/wps -dmr '
  SSH;
RSubmit;
  DATA _NULL_;
    PUT "Hello, from the remote computer";
  RUN;
ENDRSubmit;
SIGNOFF;
```

SYSRPUTSYNC

Specifies whether `%SYSRPUT` statements are actioned immediately.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSYSRPUTSYNC</code>
Option group:	<code>COMMUNICATIONS</code>
Portable	True
Restrictable	True
Saveable	True

The `%SYSRPUT` statement can be used to retrieve a macro variable from a remote host. This option determines when `%SYSRPUT` is performed.

SYSRPUTSYNC

`%SYSRPUT` statements are performed immediately.

NOSYSRPUTSYNC

`%SYSRPUT` statements are not performed until a WPS Communicate synchronisation point occurs.

The system option can be overridden by the `CSYSRPUTSYNC` option of the `RSubmit` or `SIGNON` statements.

Example

In this example, the `OPTIONS` statement is used to specify that `%SYSRPUT` is actioned only at a `CONNECT` synchronisation point.

```
OPTIONS NOSYSRPUTSYNC
```

TBUFSIZE

Specifies the buffer size for remote communication.

➤ **TBUFSIZE** ➤ = ➤ *buffer-size* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	32768
Minimum value:	0
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

buffer-size

The buffer size for remote communication. The size can be entered in bytes, in kilobytes by appending `K`, or in megabytes by appending `M`.

The system option can be overridden by the `TBUFSIZE` option of the `RSUBMIT` or `SIGNON` statements.

If you use `PROC OPTIONS` to return the value of `TBUFSIZE`, the value is the number of bytes; it does not return the value in `K` or `M` formats.

Example

In this example, the `OPTIONS` statement is used to specify that the buffer size is 32Kbytes. `PROC OPTIONS` is used to return the value you set.

```
OPTIONS TBUFSIZE = 32K;  
PROC OPTIONS OPTION=TBUFSIZE;
```

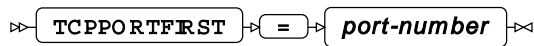
This produces the following output:

```
TBUFSIZE=32768      Buffer size for remote communication
```

The value is returned as bytes, not Kbytes.

TCPPORTFIRST

Specifies the lowest TCP/IP port number that WPS Communicate can use for the data communication channel.



Valid in:	Configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	65535
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

port-number

The TCP/IP port number.

This system option can be used with `TCPPORTLAST` to restrict the set of ports, so that different sessions have known ranges of ports.

Example

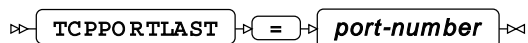
In this example, the system option is set on the command line, with the `TCPPORTLAST` also specified, to set a range of ports.

```
wps test.wps -tcpportfirst 1024 -tcpportlast 2028
```

This sets the lowest port number that can be used by WPS Communicate to 1024.

TCPPORTLAST

Specifies the highest TCP/IP port number that WPS Communicate can use for the data communication channel.



Valid in:	Configuration file and command line.
Default:	65535
Minimum value:	0

Maximum value:	65535
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

This system option can be used with `TCPPORTFIRST` to restrict the set of ports, so that different sessions have known ranges of ports.

port-number

The TCP/IP port number.

Example

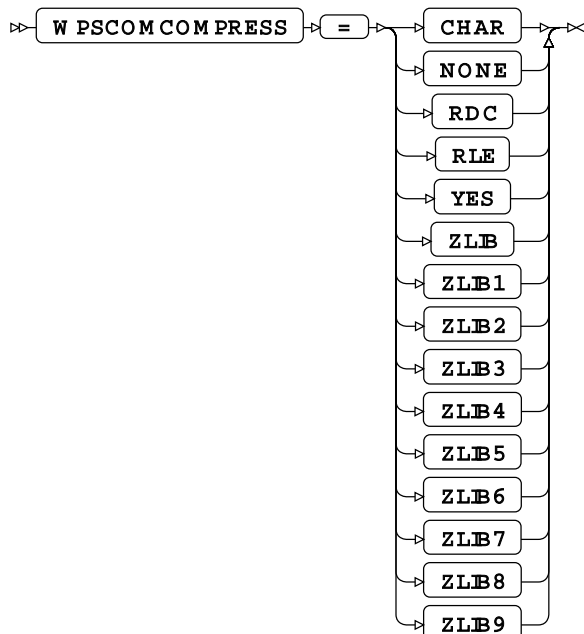
In this example, the system option is set on the command line, with the `TCPPORTFIRST` also specified, to set a range of ports.

```
wps test.wps -tcpportfirst 1024 -tcpportlast 2028
```

This sets the highest port number that can be used by WPS Communicate to 2028.

WPSCOMCOMPRESS

Specifies whether the WPS Communicate data channel uses a compression scheme.



Valid in: `OPTIONS` statement, configuration file and command line.

Default:	NONE
Maximum length:	16
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

If SSH is being used, then prefer SSH compression to WPS compression. WPS compresses all data on the data channel. Specifying this option might, therefore, decrease throughput.

CHAR

User Ross Data Compression (RDC).

NONE

Do not use a compression scheme.

RDC

User Ross Data Compression (RDC).

RLE

Run-length encoding.

YES

Use RLE encoding.

ZLIB

Use ZLIB compression. This uses the default compression (ZLIB6).

ZLIB1

Use ZLIB compression, level 1.

ZLIB2

Use ZLIB compression, level 2.

ZLIB3

Use ZLIB compression, level 3.

ZLIB4

Use ZLIB compression, level 4.

ZLIB5

Use ZLIB compression, level 5.

ZLIB6

Use ZLIB compression, level 6.

ZLIB7

Use ZLIB compression, level 7.

ZLIB8

Use ZLIB compression, level 8.

ZLIB9

Use ZLIB compression, level 9.

Example

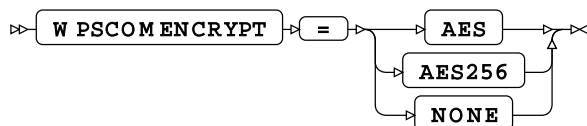
In this example, the `OPTIONS` statement is used to specify that RDC is used.

```
OPTIONS WPSCOMCOMPRESS=RDC;
SIGNON
  CONNECTREMOTE = myserver
  USERNAME = "jdoe"
  PASSWORD = "x1x2x3x4"
  SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr'
  SSH;
RSUBMIT;
  DATA _NULL_;
    PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

Communications between the local and remote servers are compressed using RDC.

WPSCOMENCRYPT

Specifies whether the WPS Communicate session data channel is encrypted.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NONE
Maximum length:	16
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

If WPS communicates with a server over an SSH connection, there is no need to use WPS encryption to encrypt the data channel. The data channel is created on a secure SSH connection. Specifying this option might, therefore, decrease throughput.

Communication with a remote server is encrypted using AES.

AES

Use basic AES encryption.

AES256

User AES-256 encryption.

NONE

Do not encrypt communications.

AES encryption requires a key, which can be specified using `WPSCOMENCRYPTKEY` [↗](#) (page 69).

Example

In this example, the `OPTIONS` statement is used to specify that basic AES encryption is used.

```
OPTIONS WPSCOMENCRYPTKEY = aaXXy1 WPSCOMENCRYPT=AES;
SIGNON
CONNECTREMOTE = myserver
USERNAME = "jdoe"
PASSWORD = "x12xx12"
SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr'
SSH;
RSUBMIT;
DATA _NULL_;
    PUT "Hello, from the remote computer";
RUN;
ENDRSUBMIT;
SIGNOFF;
```

Communication with the remote server is encrypted using AES.

WPSCOMENCRYPTKEY

Specifies the key to be used by `WPSCOMENCRYPT`.

➤ `WPSCOMENCRYPTKEY` ➤ `=` ➤ `key` ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	32000
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

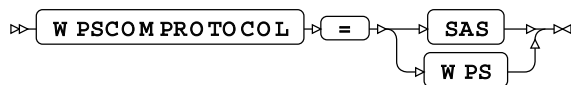
key

The key used to encrypt WPS Communicate communications. The key specified must be identical for both client and server. Encoding differences between client and server might cause the key to appear differently to each.

WPS Communicate sessions are encrypted using the `WPSCOMENCRYPT` system option.

WPSCOMPROTOCOL

Specifies the protocol used to communicate with a remote WPS server.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	WPS
Maximum length:	16
Option group:	COMMUNICATIONS
Portable	True
Restrictable	True
Saveable	True

SAS

Use the SAS Connect protocol for compatibility. This protocol might limit the facilities and capabilities available to the Communicate session.

WPS

Use the WPS protocol.

Example

In this example, the `OPTIONS` statement is used to specify that the SAS protocol is used.

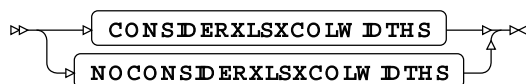
```
OPTIONS WPSCOMPROTOCOL=SAS;
SIGNON
  CONNECTREMOTE = myremote
  USERNAME = "AZ"
  PASSWORD = "*****"
  SASCMD='/opt/worldprogramming/wps-4/bin/wps -dmr'
  SSH;
RSUBMIT;
  DATA _NULL_;
    PUT "Hello, from the remote computer";
  RUN;
ENDRSUBMIT;
SIGNOFF;
```

DATABASE ENGINE group system options

The system options in this group specify settings for debug tracing for database engines.

CONSIDERXLSXCOLWIDTHS

Specifies that the width of columns defined in an Excel worksheet are used to define the length and format of variables.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOCONSIDERXLSXCOLWIDTHS`

Option group: `DATABASE ENGINE`

Portable: `True`

Restrictable: `True`

Saveable: `True`

The widths of columns in an Excel worksheet can be defined. This width can be used to define the length and formats of variables in imported datasets. This system option affects worksheets imported with both the Excel and XLSX data engines.

CONSIDERXLSXCOLWIDTHS

Use worksheet column widths to define the length and format of variables.

NOCONSIDERXLSXCOLWIDTHS

Ignore worksheet column widths. For strings, the length and format of variables is defined by the length of the longest string. The length of numbers is always eight.

If a column contains strings, and the column width is shorter than the longest string, then the length of the longest string is used as the length and format of the corresponding variable in WPS.

If a column contains numbers, and the column is shorter than the longest number, then the column width is used to specify the number format; the length of the number is always eight. For example, if the number is 10000 but the column is only three characters wide, `BEST3 .` is defined as the format. The format `BEST .` is used for all number formats, including currency and accounting. The largest format for a number is `BEST20 .`











All other Excel worksheet formats are read by WPS as described in the section *WPS Engine for Excel*. For example, a column formatted as date-time has the WPS format `DATE9 .`, whatever the column width.

Example











In this example, the `OPTIONS` statement is used to specify that column widths do not define variable lengths. An Excel worksheet is read that lists library books. The widths of the `Title` and `Author` columns have been respectively formatted as 100 characters wide and 80 characters wide. The following program reads the worksheet.

```
OPTIONS NOCONSIDERXLSXCOLWIDTHS;
LIBNAME BOOKS XLSX 'c:\temp\books\lib_books.xlsx';
DATA books_out;
    SET books.sheet1;
    OUTPUT;
RUN;
```

If you open the properties of the dataset `BOOKS_OUT` in the `WORK` library, you can see that the width of the variables `Author` and `Title` have been set to the width of the longest values for those variables, rather than the column width:

Name	Type	Length	Format	Informat	Label
 Author	character	59	\$59.	\$59.	Author
 DatePur...	number	8	DATE9.	DATE9.	DatePurchased
 Dewey_...	character	11	\$11.	\$11.	Dewey_Decimal_Number
 In_Stock	character	1	\$1.	\$1.	In_Stock
 ISBN	character	14	\$14.	\$14.	ISBN
 LastAcc...	number	8	DATE9.	DATE9.	LastAccessed
 Number...	number	8			NumberInStock
 Number...	number	8			NumberOfAccesses
 Price	number	8			Price
 Title	character	80	\$80.	\$80.	Title

If you had specified `CONSIDERXLSXCOLWIDTHS`, or left the option value at its default, then the properties for the dataset created by the `DATA` step are:

Name	Type	Length	Format	Informat	Label
 Author	character	80	\$80.	\$80.	Author
 DatePur...	number	8	DATE9.	DATE9.	DatePurchased
 Dewey_...	character	23	\$23.	\$23.	Dewey_Decimal_Number
 In_Stock	character	8	\$8.	\$8.	In_Stock
 ISBN	character	30	\$30.	\$30.	ISBN
 LastAcc...	number	8	DATE9.	DATE9.	LastAccessed
 Number...	number	8	BEST15.		NumberInStock
 Number...	number	8	BEST15.		NumberOfAccesses
 Price	number	8	BEST15.		Price
 Title	character	100	\$100.	\$100.	Title

Here, you see that the widths of the formats for `Author` and `Title` are the same as those specified in the Excel worksheet.

DBIDIRECTEXEC

Specifies whether SQL procedure statements to create or delete tables are converted to a native database query.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODBIDIRECTEXEC`

Option group: `DATABASE ENGINE`

Portable: `False`

Restrictable: `True`

Saveable: `False`

DBIDIRECTEXEC

Convert statements to a native database query.

NODBIDIRECTEXEC

Do not convert statements to a native database query.

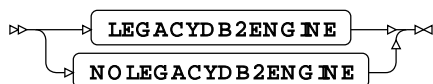
Example

In this example, the `OPTIONS` statement is used to specifying that SQL procedure statements are converted to a native database query.

```
OPTIONS = DBIDIRECTEXEC
```

LEGACYDB2ENGINE

Specifies whether the legacy DB2 data engine is used for DB2 interactions..



Valid in: Configuration file and command line.

Default: `NOLEGACYDB2ENGINE`

Option group: `DATABASE ENGINE`

Portable: `True`

Restrictable: `True`

Saveable True

LEGACYDB2ENGINE

Use the legacy engine. This is the equivalent of specifying DB2OLD as the engine to the LIBNAME statement.

NOLEGACYDB2ENGINE

Use the current version of the data engine.

Example

In this example, the system option is specified on the command line, and sets the data engine to use as the legacy version.

```
WPS c:\temp\test2.wps -LEGACYDB2ENGINE
```

LEGACYNETEZZAENGINE

Specifies whether the legacy Netezza data engine is used.



Valid in: Configuration file and command line.

Default: NOLEGACYNETEZZAENGINE

Option group: DATABASE ENGINE

Portable True

Restrictable True

Saveable True

When you specify NETEZZA as a libname, the latest version of the Netezza data engine is used. You might, however, want to use the legacy version of the data engine. This system option enables you to do this.

LEGACYNETEZZAENGINE

Use the legacy engine. This is the equivalent of specifying NETEZZAOLD as the engine to the LIBNAME statement.

NOLEGACYNETEZZAENGINE

Use the current version of the data engine.

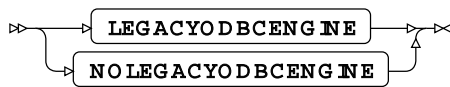
Example

In this example, the system option is specified on the command line, and sets the data engine to use as the legacy version.

```
WPS c:\temp\test2.wps -LEGACYNETEZZAENGINE
```

LEGACYODBCENGINE

Specifies whether the legacy ODBC data engine is used.



Valid in:	Configuration file and command line.
Default:	NOLEGACYODBCENGINE
Option group:	DATABASE ENGINE
Portable	True
Restrictable	True
Saveable	True

When you specify ODBC as a libname, the latest version of the ODBC data engine is used. You might, however, want to use the legacy version of the data engine. This system option enables you to do this.

LEGACYODBCENGINE

Use the legacy engine. This is the equivalent of specifying ODBCOLD as the engine to the LIBNAME statement.

NOLEGACYODBCENGINE

Use the current version of the data engine.

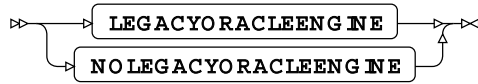
Example

In this example, the system option is specified on the command line, and sets the data engine to use as the legacy version.

```
WPS c:\temp\test2.wps -LEGACYODBCENGINE
```

LEGACYORACLEENGINE

Specifies whether the legacy Oracle data engine is used.



Valid in:	Configuration file and command line.
Default:	NOLEGACYORACLEENGINE
Option group:	DATABASE ENGINE
Portable	True
Restrictable	True
Saveable	True

When you specify `ORACLE` as a libname, the latest version of the Oracle data engine is used. You might, however, want to use the legacy version of the data engine. This system option enables you to do this.

LEGACYORACLEENGINE

Use the legacy engine. This is the equivalent of specifying `ORACLEOLD` as the engine to the `LIBNAME` statement.

NOLEGACYORACLEENGINE

Use the current version of the data engine.

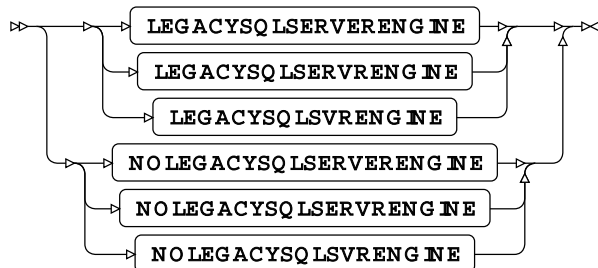
Example

In this example, the system option is specified on the command line, and sets the data engine to use as the legacy version.

```
WPS c:\temp\test2.wps -LEGACYORACLEENGINE
```

LEGACYSQLSERVERENGINE

Specifies whether the legacy SQL Server data engine is used.



Valid in:	Configuration file and command line.
Default:	NOLEGACYSQLSERVERENGINE
Option group:	DATABASE ENGINE
Portable	True
Restrictable	True
Saveable	True

When you specify `SQLSERVER` as a libname, the latest version of the SQL Server data engine is used. You might, however, want to use the legacy version of the data engine. This system option enables you to do this.

LEGACYSQLSERVERENGINE

Use the legacy engine. This is the equivalent of specifying `SQLSERVEROLD` as the engine to the `LIBNAME` statement.

NOLEGACYSQLSERVERENGINE

Use the current version of the data engine.

Example

In this example, the system option is specified on the command line, and sets the data engine to use as the legacy version.

```
WPS c:\temp\test2.wps -LEGACYSQLSERVERENGINE
```

SASTRACE

Specifies the level of debug tracing in the database engines.

» **SASTRACE** = *debug-level* «

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32
Option group:	DATABASE ENGINE
Portable	True
Restrictable	True
Saveable	True

debug-level

The level of debug tracing, must be one of:

- " , , , d "

The basic level of trace. Returns information on communications with the database, and other information at a database level.

- " , , d , "

Returns information more detailed information. It returns the information returned by " , , , d " , but adds more information at the record level.

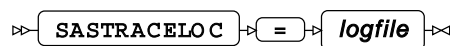
Example

In this example, the `OPTIONS` statement is used to specify the basic level of trace.

```
OPTIONS SASTRACE = ' , , d '
```

SASTRACELOC

Specifies the location of the log file to which debug tracing is written.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	1024
Option group:	<code>DATABASE ENGINE</code>
Portable	True
Restrictable	True
Saveable	True

logfile

The name of the file to which database trace debugging is written. This can be specified as an operating system path and filename, such as `c:\temp\tlog`); or as SAS-language system output file, such as `STDOUT` or `SASLOG`; or as a SAS-language filename reference or catalog.

DB2 group system options

The system options in this group specify settings for DB2 database connectivity.

DB2IN

Specifies the default DB2 tablespace in which tables are created.

➤ **DB2IN** ➤ = ➤ *db2-tablespace* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	1024
Option group:	DB2
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

db2-tablespace

The default tablespace name.

Example

In this example, the `OPTIONS` statement is used to specify that the default DB2 tablespace is used.

```
OPTIONS DB2IN = TEMPSPACE1;
```

DB2READBUFF

Specifies the number of rows to read from DB2.

➤ **DB2READBUFF** ➤ = ➤ *read-rows* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	1
Minimum value:	1
Maximum value:	32767
Option group:	DB2
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

When rows are requested from a DB2 database, this is the number of rows read and stored in memory. Larger values consume more memory as buffers are larger, but provide more read-ahead, and so can improve performance.

read-rows

The number of rows to be read.

Example

In this example, the `OPTIONS` statement is used to specify the number of rows read.

```
OPTIONS DB2READBUFF = 10000;
```

DB2SSID

Specifies the default DB2 subsystem identifier.

DB2SSID = system-ID

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	1024
Option group:	DB2
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

system-ID

The default DB2 subsystem identifier.

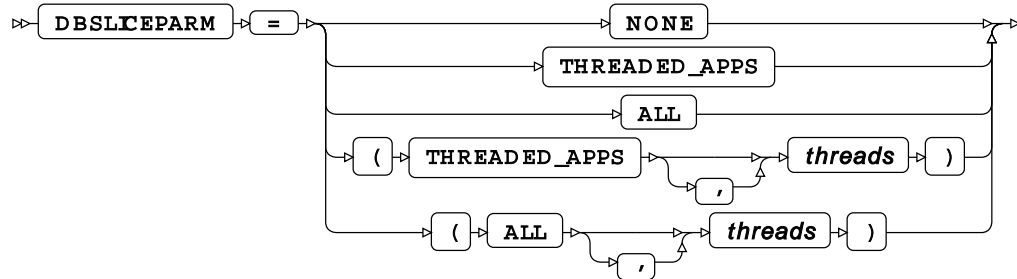
Example

In this example, the `OPTIONS` statement is used to specify the system identifier.

```
OPTIONS DB2SSID = DA1A;
```

DBSLICEPARM

Controls how WPS database connections use threaded read operations.



Valid in: OPTIONS statement, configuration file and command line.

Default: THREADED_APPS

Option group: DATABASE ENGINE
DB2

Portable: True

Restrictable: False

Saveable: True

NONE

Multiple threads are not used.

THREADED_APPS

When the keyword `THREADED_APPS` is specified, the number of available threads is equal to the number of available CPUs. The number of available threads can be limited by using the *threads* option with the `THREADED_APPS` keyword.

ALL

When only the keyword `ALL` is specified, the number of available threads is equal to the number of available CPUs. The number of available threads can be limited by using the *threads* option with the `ALL` keyword.

Example

In this example, the `OPTIONS` statement is used to specify that multiple threads are not used.

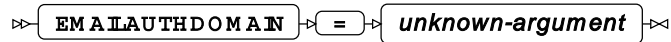
```
OPTIONS DBSLICEPARM = THREADED_APPS;
```

EMAIL group system options

The system options in this group specify settings for email connectivity. Email can be sent using the `EMAIL` access method for `FILENAME`.

EMAILAUTHDOMAIN

Specifies an authentication domain that supplies email credentials.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	80
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

You can specify an authentication domain that can be used with the `EMAIL` access method of the `FILENAME` statement. If you use an authentication domain, you might not need to specify the `EMAILPW` and `EMAILID` system options.

unknown-argument

The name of the authentication domain.

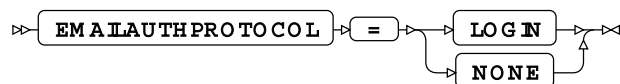
Example

In this example, the `OPTIONS` statement is used to specify that an authentication domain is used for email access.

```
OPTIONS EMAILAUTHDOMAIN=myemail.accounts;
```

EMAILAUTHPROTOCOL

Specifies whether authentication is required when WPS initiates an SMTP email connection.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NONE
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

LOGIN

Authentication is required.

NONE

Authentication is not required.

Example

In this example, the `OPTIONS` statement is used to specify authentication is required.

```
OPTIONS EMAILAUTHPROTOCOL = LOGIN;
```

EMAILHOST

Specifies the SMTP server host for the email access method.

➤ **EMAILHOST** ➤ = ➤ *server-id* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	localhost
Maximum length:	1024
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

server-id

The SMTP server name or IP address to use. By default this the host name `localhost`.

Example

In this example, the `OPTIONS` statement is used to specify the address of the SMTP server.

```
OPTIONS EMAILHOST = smtp.ourserver.com;
```

EMAILID

Specifies the identifier used to authenticate a user when connecting to the SMTP server.

➤ **EMAILID** ➤ = ➤ *SMTP-login-id* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32000
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

SMTP-login-id

The user identifier. If the identifier contains spaces, you must enclose it in quotation marks.

Example

In this example, the `OPTIONS` statement is used to specify the user identifier.

```
OPTIONS EMAILID = 'SMTP User 1';
```

EMAILMASQUERADEHOST

Specifies a fully qualified domain name to masquerade as your email domain name.

```
» EMAILMASQUERADEHOST = fdqn «
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	1024
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

An email masquerade enables you to set the server component of the outgoing email address for sent SMTP emails, instead of the actual host name of the sending server. The name specified is used instead of, or in place of, the name specified in the `FROM` option of the `EMAIL` access method of `FILENAME`. For example, you might want email that would normally appear with the host name `mybigcompany.com` to `madeupco.co.uk`.

fdqn

The fully qualified domain name of the masquerading email address.

Example

In this example, the `OPTIONS` statement is used to specify the email masquerade address.

```
OPTIONS EMAILMASQUERADEHOST = madeupco.co.uk;
```

EMAILPORT

Specifies the port number of the SMTP server for the email access method.

⇒ **EMAILPORT** ⇒ = ⇒ *port-number* ⇒ ⇐

Valid in:	OPTIONS statement, configuration file and command line.
Default:	25
Minimum value:	0
Maximum value:	2147483647
Option group:	EMAIL
Portable	False
Restrictable	True
Saveable	False

port-number

The port number for the SMTP server.

Example

In this example, the `OPTIONS` statement is used to set the port number used by the SMTP server to 125.

```
OPTIONS EMAILPORT = 125
```

EMAILPW

Specifies the password to use when authentication is required to connect to an SMTP server.

⇒ **EMAILPW** ⇒ = ⇒ *SMTP-login-passwd* ⇒ ⇐

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32000
Option group:	EMAIL

Portable	False
Restrictable	True
Saveable	False

SMTP-login-passwd

The password for the identifier specified in the `EMAILID` [↗](#) (page 83) system option. If the password contains spaces, you must enclose it in quotes.

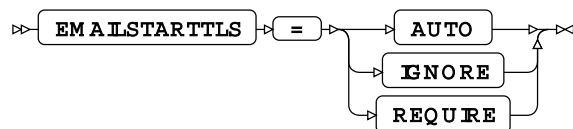
Example

In this example, the `OPTIONS` statement is used to specify the password for an email.

```
OPTIONS EMAILHOST = mysimplepassword;
```

EMAILSTARTTLS

Specifies whether `STARTTLS` is used to secure SMTP email connections.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>AUTO</code>
Option group:	<code>EMAIL</code>
Portable	False
Restrictable	True
Saveable	False

AUTO

If possible, the SMTP communication channel is secured using TLS encryption before email is sent over it. Otherwise, the email is sent over an unencrypted channel.

IGNORE

The email is sent over an unencrypted channel. No start TLS negotiation is performed. SMTP mail transfer fails if the server requires `STARTTLS`.

REQUIRE

The SMTP communication channel must be secured using TLS encryption before email is sent over it. If it is not, an error is generated and the email is not sent. the SMTP server must support `STARTTLS`.

`STARTTLS` is a mechanism that changes a plain text connection to an encrypted connection.

Example

In this example, the `OPTIONS` statement is used to specify that `STARTTLS` is ignored.

```
OPTIONS EMAILSTARTTLS = IGNORE;
```

EMAILSYS

Specifies the type of email system.

⇒ **EMAILSYS** ⇒ = ⇒ ***interface-name*** ⇒

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>MAPI</code>
Option group:	<code>EMAIL</code>
Portable	False
Restrictable	True
Saveable	False

interface-name

Can be one of :

- `CSSMTP`
- `MAPI`
- `SMTP`

On Windows operating systems, the default value is `MAPI`; on all other operating systems the default value is `SMTP`.

Example

In this example, the `OPTIONS` statement is used to set the type of email system to `SMTP`.

```
OPTIONS EMAILSYS = SMTP;
```

ENVDISPLAY group system options

The system options in this group define display characteristics.

CHARCODE

Specifies whether character combinations can be used as a substitute for special characters that are not present on the keyboard.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOCHARCODE
Option group:	ENVDISPLAY
Portable	True
Restrictable	True
Saveable	True

CHARCODE

Enable character combinations.

NOCHARCODE

Do not enable character combinations.

This system option enables you to enter characters that are not on the keyboard by using combinations of characters. If this system option is set, all such character combinations are replaced by the equivalent character.

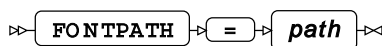
Example

In this example, the `OPTIONS` statement is used to specify that character combinations cannot be used.

```
OPTIONS NOCHARCODE;
```

FONTPATH

Specifies search paths to use when locating TrueType fonts.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	2048
Option group:	ENVDISPLAY
Portable	False
Restrictable	True

Saveable False

path

List of paths to search for TrueType fonts. More than one path can be specified. To specify multiple paths, use the format ('*path1*' '*path2*' ...). See the example below.

Example

In this example, the `OPTIONS` statement is used to specify a Windows folder containing TrueType fonts:

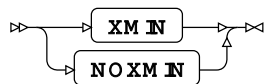
```
OPTIONS FONTPATH = c:\work\fonts;
```

In this example, the `OPTIONS` statement is used to specify multiple Windows folders containing TrueType fonts:

```
OPTIONS FONTPATH = ('c:\work\fonts' 'n:\all\fonts');
```

XMIN

Specifies whether windows that are opened by commands executed using the `x` statement, the `SYSTEM` function, or the `CALL SYSTEM` routine are minimised on starting.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOXMIN</code>
Option group:	<code>ENVDISPLAY</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	64-bit Windows 32-bit Windows

Only the window opened by the specified command is minimised. If a process executed by the command also opens a window, that window is not minimised. For example, if this system option is set, and you start Windows Notepad directly using `x`, the Notepad window is started in its minimized state; if you start Notepad from the Windows Command Prompt window, then the Command Prompt window is minimised, but the Notepad window is not.

XMIN

Windows are minimized.

NOXMIN

Windows are not minimized.

Basic example

In this example, the `OPTIONS` statement is used to specify that the operating system command is run in a default, non-minimised window.

```
OPTIONS NOXMIN;  
X dir c:\temp;
```

Example - starting application

In this example, the `OPTIONS` statement is used to specify that an application starts in a minimised Window.

```
OPTIONS XMIN;  
DATA _NULL_;  
    CALL SYSTEM("notepad");  
RUN;
```

The Windows Notepad application starts minimised.

Example - starting application from command prompt

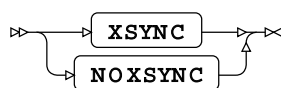
In this example, the `OPTIONS` statement is used to specify that the Command Prompt window is started minimized.

```
OPTIONS XMIN;  
DATA _NULL_;  
    CALL SYSTEM("start notepad");  
RUN;
```

Specifying the Windows `START` command first opens the Windows Command Prompt, and applies the command to start Notepad in that prompt. The Command Prompt window starts minimised, the Notepad window does not.

XSYNC

Specifies whether to wait for applications or commands, launched through the `X` statement, the `SYSTEM` function, or the `CALL SYSTEM` routine, to finish before continuing program execution.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOXSYNC`

Option group:	ENVDISPLAY
Portable	False
Restrictable	True
Saveable	True
Supported platform:	64-bit Windows 32-bit Windows

An application or command is considered to have finished when the Windows Command Prompt window is closed, either by the user or by the application or program.

XSYNC

WPS waits before continuing.

NOXSYNC

WPS does not wait and continues program execution.

Example

In this example, the `OPTIONS` statement is used to specify that the program does not continue to execute until applications and commands have finished.

```
OPTIONS XSYNC;  
DATA _NULL_;  
    rc = SYSTEM('dir c:');  
RUN;
```

The program executes the Windows `DIR` command in a command window. The SAS language program stops executing until the command window is closed.

The log shows:

```
real time : 2.272  
cpu time  : 0.015
```

The CPU time reflects the time taken to execute the SAS language program. The amount of real time elapsed also includes the amount of time taken to check the command window and then close it.

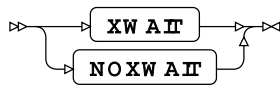
If the system option had been set as `OPTIONS NOXSYNC`, the log might show something similar to:

```
real time : 0.017  
cpu time  : 0.015
```

This time, both figures are roughly the same, as the program did not wait for the command window before continuing.

XWAIT

Specifies that a Command Prompt opened using the `X` statement, the `SYSTEM` function or the `CALL SYSTEM` routine, remains active until you exit from it.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOXWAIT
Option group:	ENVDISPLAY
Portable	False
Restrictable	True
Saveable	True
Supported platform:	64-bit Windows 32-bit Windows

XWAIT

The Command Prompt window waits for you exit from it.

NOXWAIT

The Command Prompt window does not wait for you to exit from it.

A Windows command can be started from a SAS language program using the `X` statement, the `SYSTEM` function or a `CALL SYSTEM` routine.

If a command opens a Command Prompt window, then:

- If `XWAIT` is set, the window remains open until closed
- If `NOXWAIT` is set, the window only remains open while the command executes.

The SAS language program that invoked the command runs to completion, unless the `XSYNC` [link](#) (page 90) system option is set.

Note:

Windows PowerShell has its own option that controls the display of its command window, and is unaffected by the setting of this system option.

Example

In this example, the `OPTIONS` statement is used to specify that the Command Prompt window remains open after the Windows command has executed.

```
OPTIONS XWAIT;
DATA _NULL_;
  rc = SYSTEM('dir c:');
RUN;
```

The program executes the Windows `DIR` command in a Command Prompt window. The window remains open until it is closed.

If the system option had been set as `OPTIONS NOWAIT`, the Command Prompt window would have opened very briefly while the command executed, and then closed when the Windows command finished.

ENVFILES group system options

The system options in this group specify environment settings for files.

ALTLOG

Specifies that a duplicate of the WPS server log is created, and the file name for the duplicate log.

➤ **ALTLOG** ➤ = ➤ *filepath* ➤

Valid in:	Configuration file and command line.
Maximum length:	256
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	False

All text written to the WPS server log is automatically copied to the alternative log in the specified location. Log information from multiple WPS servers cannot be written to the same alternative log file; you must define an alternative log location for each WPS server. This system option can be restricted for additional security.

filepath

Specifies the filename for the alternative log file.

The file name can contain naming directives to control the name format. To use the naming directives, you must specify `rollover=auto` or `rollover=session` in the `LOGPARM` system option.

Note:

If the filename contains naming directives, and the `LOGPARM` [↗](#) (page 241) system option is set to overwrite the existing log (that is, the `REPLACE` or `REPLACEOLD` option of `LOGPARM` has been set), then the `ROLLOVER` option of `LOGPARM` [↗](#) (page 241) must be set to `AUTO`.

Each directive must be preceded by an escape character, either %, or #. On z/OS platforms, you must use the # escape character.

The literal escape characters can be included in the file by preceding them with another escape character. For example, ## outputs # as part of the file name.

The supported naming directives are:

- A Adds the full name of the day (for example, Sunday) to the file name.
- a Adds the abbreviated name of the day (for example, Sun) to the file name.
- B Adds the full month name (for example, January) to the file name.
- b Adds the abbreviated month name (for example, Jan) to the file name.
- C Adds the numeric century value to the file name.
- d Adds the numeric day of the month to the file name.
- H Adds the hour at which the log was created to the file name.
- j Add the numeric Julian day value to the file name.
- l Adds the login name to the file name.
- m Adds the ordinal numeric month value to the file name, starting at 1 for January to 12 for December.
- M Adds the minute at which the log was created to the file name.
- n Adds the unqualified host name of the WPS server host machine to the file name
- p Adds the process ID of the WPS server to the file name, enabling you to identify individual WPS servers where multiple servers run on the same host machine.
- s Adds the second at which the log was created to the file name. The log will not roll over more frequently than once per minute, but will roll over at the first minute boundary after the log is created.

For example, if the first log is created at 9:00:47, the second is created at 9:01:00, the third at 9:02:00, and so on.

- u Adds the ordinal numeric day of the week in the file name, starting at 1 for Monday to 7 for Sunday.
- v Adds a unique identifier to the file name in the form:

```
<proces-id>v<counter>
```

The counter is incremented until a unique file name can be created. This directive can only be used once per file name definition.

- W Adds the week number in the year to the file name. The first day of the week is Monday, and week number one begins on the first Monday of the year. Any days in the year before the first Monday are treated as week 0 (zero).
- w Adds the numeric day of the week to the file name, starting at 0 (zero) for Sunday to 6 for Saturday.
- Y Adds the four-digit year to the file name.

- y Adds the two-digit year without the century (for example, 08 for 2008) to the file name.

If you use Workbench and specify `ALTLOG` in a configuration file, to ensure each WPS server writes information to a different alternative log file you must:

- Specify either the `#v`, or `#p` directives in the `ALTLOG` file name, and
- Specify `rollover=auto` or `rollover=session` in the `LOGPARM` system option.

Basic example

In this example, the system option is specified on the command line, and provides a name and location for the WPS log.

```
wps tscript.wps -log c:\temp\op.log -altlog m:\logs\op-alt.log
```

This directs WPS to write the log output to the file `op.log` in the folder `c:\temp`, and to the alternative log file `op-alt.log` in the folder `m:\logs`.

Example – example with filename substitutions

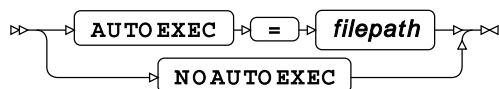
In this example, the system option is specified on the command line, and provides a name and location for the WPS log.

```
wps tscript.wps -logparm "rollover = auto"
-log c:\temp\log%u%H%M.log -altlog m:\logs\log%u%H%M.log
```

This directs WPS to write the log output in the file `log21125.log` in the folder `c:\temp`. In this filename, various substitutions have been made reflecting the time at which the log was created. `%u` has been replaced by 2 (the second day of the week, Tuesday); `%H` has been replaced by the hour of the day (11) and `%M` has been replaced by the minutes past the hour (25). A file with the same name has been written to the alternative log location, `m:\logs`.

AUTOEXEC

Specifies whether a file is automatically executed when WPS starts.



Valid in:	Configuration file and command line.
Maximum length:	1024
Option group:	ENVFILES
Appendable	True
Portable	False
Restrictable	False

Saveable False

AUTOEXEC

Specifies the name of one or more files to be automatically executed. If more than one filename is specified, surround the filenames with brackets.

NOAUTOEXEC

No file is automatically executed.

The statements in the program that is autoexecuted are not listed in the log unless the `ECHOAUTO` [↗](#) (page 235) system option is set.

Basic Example

In this example, the system option is specified on the command line, and names a program that is executed when WPS starts.

```
WPS c:\temp\test2.wps -AUTOEXEC c:\temp\test.wps
```

`test.wps` is executed first, and then `test2.wps` is executed.

The results are written to the log.

```
NOTE: AUTOEXEC processing beginning; file is c:\temp\test.wps

This program runs first
NOTE: The data step took :
      real time : 0.013
      cpu time  : 0.000

NOTE: AUTOEXEC processing completed

1      DATA _NULL_;
2      PUT 'This program runs second';
3      RUN;

This program runs second
NOTE: The data step took :
      real time : 0.029
      cpu time  : 0.015

NOTE: Submitted statements took :
      real time : 0.254
      cpu time  : 0.156
```

The statements in the program that is autoexecuted are not listed in the log. If you want list the statements in the autoexecuted programs, specify the `ECHOAUTO` system option.

Example – specifying more than one program

In this example, the system option is specified on the command line, and names two programs that are executed when WPS starts.

```
WPS c:\temp\noo.wps -AUTOEXEC "(c:\temp\tscript3.wps c:\temp\tscript2.wps) "
```

tscript3.wps is executed first, and then tscript2.wps; finally noo.wps is executed. The argument to -AUTOEXEC has been specified in quotation marks, because on the Windows command line, the space between the filenames would otherwise be regarded as a separator between arguments.

The results are written to the log.

```
NOTE: AUTOEXEC processing beginning: (c:\temp\tscript3.wps c:\temp\tscript2.wps)
NOTE: AUTOEXEC processing: file c:\temp\tscript3.wps

This starts second
NOTE: The data step took :
      real time : 0.002
      cpu time  : 0.000

NOTE: AUTOEXEC processing: file c:\temp\tscript2.wps

This starts second
NOTE: The data step took :
      real time : 0.002
      cpu time  : 0.000

NOTE: AUTOEXEC processing completed

1          DATA _NULL_;
2
3          PUT 'Hello there!';
4
5          RUN;

Hello there!
NOTE: The data step took :
      real time : 0.007
      cpu time  : 0.000

6
7
8
9

NOTE: Submitted statements took :
      real time : 0.089
      cpu time  : 0.046
```

The statements in the programs that are autoexecuted are not listed in the log. If you want list the statements in the autoexecuted programs, specify the ECHOAUTO system option.

FMTSEARCH

Specifies the catalogs to search to locate user formats.

⇒ **FMTSEARCH** ⇒ = ⇒ (⇒ *path* ⇒) ⇒ ⇐

Valid in:	OPTIONS statement, configuration file and command line.
Default:	()
Maximum length:	2048
Option group:	ENVFILES
Appendable	True
Portable	True
Restrictable	True
Saveable	True

path

One or more catalogs. The catalogs must be enclosed in parentheses.

The paths specified must be two-part names, providing the library name and the catalog name. If you do not provide a catalog name, it defaults to `FORMATS`. For example, if you specify `FMTSEARCH = (WORK)`, the catalog `FORMAT` in the temporary library `WORK` is searched for formats.

Example

In this example, the `OPTIONS` statement is used to specify the path for the user formats.

```
LIBNAME books "c:\temp\books";  
LIBNAME olib "c:\newfmt";  
OPTIONS FMTSEARCH = (work, books.sformats, olib);
```

WPS searches for user formats in the catalog `FORMATS` in the libraries `work` and `olib`, and in the catalog `SFORMATS` in the library `books`.

LOG

Specifies the path and filename for the WPS log.

⇒ **LOG** ⇒ = ⇒ *filepath* ⇒ ⇐

Valid in:	Configuration file and command line.
Maximum length:	256
Option group:	ENVFILES

Portable	True
Restrictable	False
Saveable	False

If the specified file already exists, it is overwritten unless you specify `LOGPARM` [\(page 241\)](#), which enables you to set various parameters that control what happens to the log file.

filepath

The path and filename.

The log file name can contain naming directives that control the name format for the log file. Each directive must be preceded by an escape character, either %, or #. On z/OS platforms, you must use the # escape character.

Note:

If the filename contains naming directives, and the `LOGPARM` [\(page 241\)](#) system option is set to overwrite the existing log (that is, the `REPLACE` or `REPLACEOLD` option of `LOGPARM` has been set), then the `ROLLOVER` option of `LOGPARM` [\(page 241\)](#) must be set to `AUTO`.

Escape characters can be included in the log file name as literal characters by preceding them with another escape character. For example, ## outputs # as part of the log file name.

The supported naming directives are:

- A Adds the full name of the day (for example, Sunday) to the log file name.
- a Adds the abbreviated name of the day (for example, Sun) to the log file name.
- B Adds the full month name (for example, January) to the log file name.
- b Adds the abbreviated month name (for example, Jan) to the log file name.
- c Adds the numeric century value to the log file name.
- d Adds the numeric day of the month to the log file name.
- H Adds the hour at which the log was created to the log file name.
- j Add the numeric Julian day value to the log file name.
- l Adds the user name to the log file name.
- M Adds the minute at which the log was created to the log file name.
- m Adds the ordinal numeric month value to the log file name, starting at 1 for January to 12 for December.
- n Adds the unqualified host name of the WPS server host machine to the log file name
- p Adds the process ID of the WPS server to the log file name, enabling you to identify individual WPS servers where multiple servers run on the same host machine.
- s Adds the second at which the log was created to the log file name. The log will not roll over more frequently than once per minute, but will roll over at the first minute boundary after the log is created.

For example, if the first log is created at 9:00:47, the second is created at 9:01:00, the third at 9:02:00, and so on.

- u Adds the ordinal numeric day of the week in the log file name, starting at 1 for Monday to 7 for Sunday.

- v Adds a unique identifier to the log file name in the form:

```
process-idvcounter
```

The counter is incremented until a unique file name can be created. This directive can only be used once per file name definition.

- W Adds the week number in the year to the log file name. The first day of the week is Monday, and week number one begins on the first Monday of the year. Any days in the year before the first Monday are treated as week 0 (zero).
- w Adds the numeric day of the week to the log file name, starting at 0 (zero) for Sunday to 6 for Saturday.
- Y Adds the four-digit year to the log file name.
- y Adds the two-digit year without the century (for example, 08 for 2008) to the file name.

If you do not specify this system option, the log file is written to a default location. On z/OS, the default location is the DD `SASLOG`. On Windows, the default is the console output. On other platforms, the default is a filename formed from the name of the input program with the extension replaced by the text `log`; for example, for a program `myprog.wps`, the log file is `myprog.log` in the same location as the program

Basic example

In this example, the system option is specified on the command line, and provides a name and location for the WPS log.

```
wps c:\temp\test2.wps -log c:\temp\op.log
```

This directs WPS to write the log output in the file `op.log` in the folder `c:\temp`.

Example – example with filename substitutions

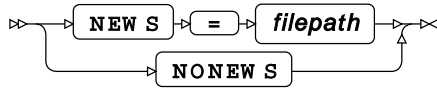
In this example, the system option is specified on the command line, and provides a name and location for the WPS log.

```
wps tscript.wps -logparm "rollover = auto" -log c:\temp\log%u%H%M.log
```

This directs WPS to write the log output in the file `log21125.log` in the folder `c:\temp`. In this filename, various substitutions have been made reflecting the time at which the log was created. `%u` has been replaced by 2 (the second day of the week, Tuesday); `%H` has been replaced by the hour of the day (11) and `%M` has been replaced by the minutes past the hour (25).

NEWS

Specifies the name of a file that contains messages to be written at the top of the log output.



Valid in:	Configuration file and command line.
Default:	<empty-string>
Maximum length:	255
Option group:	ENVFILES LOGCONTROL
Portable	True
Restrictable	True
Saveable	False

NEWS

Messages from the specified *filename* are written to the log.

NONEWS

News messages are not written to the log.

The messages are written at the top of the log, after information about WPS Analytics.

Example

In this example, the `NEWS` system option is specified on the command line. The file specified by the system option contains a message. The results are written to the log.

```
wps c:\temp\sadd.wps -news c:\temp\logmsg.txt
```

The file `logmsg.txt` contains the following:

```
Good day.
Remember to lock your screen when you leave your desk.
```

This produces the following output:

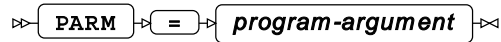
```
NOTE: (c) Copyright World Programming Limited 2002-2019. All rights reserved.
NOTE: World Programming System 4.01 (04.01.00.00.014673)
      Licensed to World Programming Company Ltd, 80 installations
NOTE: This session is executing on the X64_WIN10PRO platform and is running in 64
      bit mode

Good day.
Remember to lock your screen when you leave your desk.
```

Subsequent lines of the log contain program execution information, as usual.

PARM

Specifies a parameter string to pass to external programs.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	2048
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	True

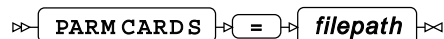
program-argument

The parameter string to pass to external programs.

This system option only has an affect when WPS is used on z/OS. When WPS is used with an unknown procedure is assumed to be an external program. For example, WPS would assume `PROC FOO` to be a reference to an external program `FOO`. This system option enables you to specify parameters that are passed when the external program is invoked. The format of the parameters depend on the external program. Parameters are separated by spaces.

PARMCARDS

Specifies the name of a file to use as the `PARMCARDS` file.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	True

filepath

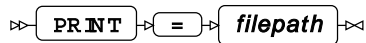
The name and path of the file to use.

This only has an affect when WPS is used on z/OS. Any unknown procedure is assumed to be an external program. For example, WPS would assume `PROC FOO` to be a reference to an external

program `FOO`. This system option enables you to specify a file containing parameters to pass to the external program. The format of the parameters depend on the external program.

PRINT

Specifies the name of a file to which the listing output is written.



Valid in:	Configuration file and command line.
Maximum length:	1024
Option group:	ENVFILES
Portable	False
Restrictable	False
Saveable	False

filepath

The path and name of the listing output file.

Example

In this example, the system option is specified on the command line, and names the file to which the listing is written. The following `DATA` step creates two variables:

```
DATA _NULL_;  
  x = 2 + 1;  
  y = 3;  
  FILE PRINT ODS;  
  PUT x y;  
RUN;
```

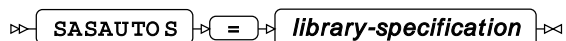
The `FILE PRINT ODS` statement directs output to a listing file. The command:

```
wps c:\temp\sadd.wps -print c:\temp\outlst.lst
```

runs the program and writes the listing output to the file `outlst.lst`, which is created in the specified folder.

SASAUTOS

Specifies the list of locations to be searched for autocall macros.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	2048
Option group:	ENVFILES MACRO
Appendable	True
Portable	True
Restrictable	True
Saveable	True

This system option provides a list of locations to be searched for unknown macros encountered in SAS language programs. If an unknown macro is found in a source program, the specified locations are searched for source files with the name of that macro. If a suitable source file is found it is read and processed as if it had been included in the source program with the %INCLUDE statement.

You must also specify the MAUTOSOURCE [↗](#) (page 103) system option.

The source file does not have to contain a macro of the same name. The source file might instead contain only non-macro statements. In this case, a message is written to the log noting that the autocall member did not contain a macro of the same name. If you run the macro again, an error message is returned, unless you also set the MRECALL [↗](#) (page 277) system option.

library-specification

One or more locations to be searched. An autocall location can be specified as an operating system pathname, or as an existing filename reference, or as an external DD card on z/OS. If multiple locations are specified, enter them in parentheses. For example, to specify one autocall location `SASAUTOS = location-specification`; to specify two autocall locations, `SASAUTOS = (location-specification1 location-specification2)`

Enter an operating system pathname inside quotation marks.

The length of *library-specification* can only be 2048 bytes or less.

Example

In this example, the OPTIONS statement is used to specify that there are two autocall locations. The SASAUTOS option must be used with the MAUTOSOURCE system option.

```
FILENAME ms1 'c:\temp\macros';  
OPTIONS MAUTOSOURCE SASAUTOS=(ms1 'c:\temp\macros2')
```

If one location contained the file `test.wps`, and the other the file `test2.wps`, you could compile and run the macros in these files by executing:

```
%test;  
%test2;
```

SASHELP

Specifies the location of the SASHELP library.

➤ **SASHELP** ➤ = ➤ *library-reference* ➤

Valid in:	Configuration file and command line.
Maximum length:	8192
Option group:	ENVFILES
Appendable	True
Portable	True
Restrictable	True
Saveable	False

library-reference

The pathname of the library location.

The SASHELP library is a standard library that is installed when you install WPS, and contains information that controls features of a WPS session. By default, the library is in the installation location. For example, on Windows, WPS is by default installed in C:\Program Files\World Programming\WPS\version, where *version* is the version number of WPS. You can, however, move or copy the SASHELP library to another location.

The SASHELP library contains datasets that you might want to customise, such as the MIME type dataset. If you customise such a dataset, you can save it in a local location rather than the SASHELP library; the local location can then be specified to this system option by concatenating it with the SASHELP library location, using the format:

```
SASHELP = ('local-lib-path' 'installation-sashelp')
```

where *local-lib-path* is the path to the local library, and *installation-sashelp* is the path to the installed SASHELP library.

Basic example

In this example, the system option is specified on the command line, and specifies the location of SASHELP when WPS starts.

```
wps c:\temp\test.wps -sashelp c:\temp\sashlpdir
```

Example – concatenating locations

In this example, the system option is specified on the command line, and specifies the location of SASHELP and a local folder that is checked for files before SASHELP.

```
wps c:\temp\test.wps -sashelp "('c:\temp\localdir'  
    'c:\Program Files\World Programming\WPS\4\sashelp')"
```

SASINITIALFOLDER

Specifies the current working directory for the WPS server.

➤ **SASINITIALFOLDER** ➤ = ➤ *filepath* ➤

Valid in:	Configuration file and command line.
Maximum length:	260
Option group:	ENVFILES
Portable	False
Restrictable	True
Saveable	False
Supported platform:	AIX for pSeries 64-bit Linux for ARM Linux for pSeries Linux (LE) for pSeries Linux for System z 64-bit Linux for System z 64-bit Linux 32-bit Linux 64-bit Mac O/S Solararis for SPARC Solaris for 64-bit x86 Solaris for 32-bit x86 64-bit Windows 32-bit Windows

filepath

The path and filename.

Example

In this example, the system option is specified on the command line, and sets the current working directory. The option **AUTOEXEC** is also specified, so an additional program is run. This program is in the folder specified by **SASINITIALFOLDER**.

```
wps c:\temp\test.wps -sasinitialfolder c:\temp\sastmp -autoexec test2.wps
```

test2.wps is executed first, and then *c:\temp\test.wps*.

SASUSER

Specifies the location of the SASUSER library.

➤ **SASUSER** ➤ = ➤ *library-reference* ➤

Valid in:	Configuration file and command line.
Maximum length:	8192
Option group:	ENVFILES
Portable	True
Restrictable	False
Saveable	False

library-reference

The path to the library location.

The **SASUSER** library contains various files specific to a user. The library is created in a default location when WPS is installed; for example, on Windows, it is the folder `Documents/My WPS Files` for a particular user. You can specify a different location for the library, if required.

Example

In this example, the system option is specified on the command line, and set the path of **SASUSER**.

```
WPS c:\temp\test2.wps -SASUSER c:\temp\sastmp
```

SET

Enables you to set the values of environment variables.

➤ **SET** ➤ = ➤ *variable-name* ➤ " ➤ *variable-value* ➤ " ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32000
Option group:	ENVFILES
Portable	False
Restrictable	True
Saveable	True

Environment variables are variables that can be set for a WPS session, and used by SAS language programs during that session. This enables you to set an item such as a directory pathname once at the beginning of the session, and then use that item in one or more programs during the session.

variable-name

The name of the environment variable. This is a name you specify that can then be used by programs during a session.

variable-value

The value for the environment variable. This must be a quoted string. This value can include a WPS environment variable, a system environment variable, or a Windows constant special item ID list (CSIDL) value. To include a WPS environment variable or system environment variable, prefix the string with a `!` (exclamation mark). To include a CSIDL value, prefix the string with a `?` (question mark).

For example, `SET = tmp 'c:\temp'` sets a system variable `tmp` to the specified path; `SET = tmp '!HOMEPATH'` sets a system variable `tmp` to the specified Windows environment variable, and therefore to the path specified by that environment variable.

If you specify this system option to the `OPTIONS` procedure, all environment variables set using this option in this WPS session are listed, including any set in configuration files when the session started.

Basic example

In this example, the `OPTIONS` statement is used to specify a system variable.

```
OPTION SET = tmp "c:\temp";
FILENAME test '!tmp\logmsg.txt';
DATA _NULL_;
  INFILE test;
  INPUT;
  PUT _INFILE_;
RUN;
```

The system variable `tmp` has been set to `c:\temp`. If there is a file `logmsg.txt` in the folder `c:\temp`, the observations in it are read and written to the log.

If you were to then specify `SET` to the `OPTIONS` procedure, the `tmp` variable is listed as an environment variable, in addition to any options set by configuration files. For example:

```
PROC OPTIONS OPTION = SET;
```

This writes the following to the log:

```
Environment Variables:

SASAUTOS=( '!wpshome\sasmacro' )
tmp=c:\temp
```

Example – using operating system environment variables

In this example, the `OPTIONS` statement is used to specify a system variable that has the value of an operating system environment variable.

```
OPTIONS SET = tmp "?CSIDL_DESKTOP";
FILENAME test '!tmp\logmsg.txt';
DATA _NULL_;
    INFILE test;
    INPUT;
    PUT _INFILE_;
RUN;
```

The system variable `tmp` has been set to the CSIDL value `CSIDL_DESKTOP`. This identifies the folder used to store Windows desktop items. If there is a file `logmsg.txt` on the Windows desktop, the observations in it are read and written to the log. The CSIDL requires that you use the special character `?` to identify it.

In this example, if you specify `SET` to the `OPTIONS` procedure, the `tmp` variable is listed as an environment variable, in addition to any options set by configuration files. For example:

```
PROC OPTIONS OPTION = SET;
```

This writes the following to the log:

```
Environment Variables:

SASAUTOS=( '!wpshome\sasmacro' )
tmp=!CSIDL_DESKTOP
```

In the log, the CSIDL value is prefixed with an exclamation mark.

SYSIN

Specifies the program file from which source code is read.

```
» SYSIN = filepath «
```

Valid in:	Configuration file and command line.
Maximum length:	1024
Option group:	ENVFILES
Portable	False
Restrictable	False
Saveable	False

filepath

The program file. Also include the pathname of the file if required. On z/OS, this can be a name specified by a DD statement.

On z/OS, you can specify `SYSDIN=SASIN` to read program source from `SASIN DD`, rather than from the default `SYSDIN DD`.

Example

In this example, the system option is specified on the command line, and specifies the file containing the program to be executed.

```
WPS -SYSDIN c:\temp\tscript.wps
```

SYSPARM

Specifies a character string that is used to initialise the `SYSPARM` macro variable.

➤ **SYSPARM** ➤ = ➤ *character-value* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32767
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	True

character-value

The string to be passed to `SYSPARM`.

This system option could be used, for example, to pass information from the command line to the program; in this case, you specify the value on the command line. That value might itself be included in the command line using some operating system method.

To access the information in a `DATA` step, use the `SYSPARM` function, which returns a character string. To access the information as a macro variable, use the automatic macro variable `&SYSPARM`.

Basic example

In this example, the `OPTIONS` statement is used to specify the value passed to the `SYSPARM` variable.

```
OPTIONS SYSPARM="Bicycle";  
%PUT &SYSPARM;
```

`Bicycle` is written to the log.

Example – specifying value on the command line

In this example, the option is specified on the command line. The value is then passed to the program.

```
WPS -sysparm='Bye' m:\wps_test_progs\tscript.wps
```

Farewell! is written to the log.

The program is:

```
DATA _NULL_;  
  IF SYSPARM() EQ 'Hello' THEN PUT 'Welcome!';  
  IF SYSPARM() EQ 'Bye' THEN PUT 'Farewell!';  
RUN;
```

USER

Specifies the default location for all one-level dataset names.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	1024
Option group:	ENVFILES
Portable	True
Restrictable	False
Saveable	True

library-reference

The location to be used. It can be specified as a WPS library reference or as an operating system pathname. If it is a folder or directory, it must exist before use – the directory is not automatically created on access.

A message is displayed if the directory or folder does not exist:

```
NOTE: Library USER does not exist
```

Example

In this example, the `OPTIONS` statement is used to specify that `c:\temp` is used as the default library.

```
OPTIONS USER="c:\temp";
```

You could then run a program:

```
DATA out1;  
  ovar = 1;  
RUN;
```

If the system option `USER` had not been set, the dataset `out1` would have been written in the `WORK` library. However, `USER` has been set, so the `out1` is written to the specified folder.

WORK

Specifies the location of the `WORK` library used for temporary datasets and catalogs.

`WORK` = `location`

Valid in:	Configuration file and command line.
Maximum length:	1024
Option group:	ENVFILES
Portable	True
Restrictable	False
Saveable	False

Each WPS session creates a folder or directory that is used for temporary datasets and catalogs. These folders or libraries are created in the `WORK` library. A `WORK` library location is specified by default when you install WPS. For example, on Windows, it is `C:\Users\user-name\AppData\Local\Temp\WPS Temporary Data`; on Linux, it is `/tmp`.

This system option enables you to set your own location for the `WORK` library.

location

The path to the `WORK` library location.

If the location does not exist, it is created.

Folders in the `WORK` library location are deleted at the end of the WPS session. If you want to keep the working datasets you have created, you must also set `NOWORKTERM`. However, although you can use `NOWORKINIT` [↗](#) (page 113) and `NOWORKTERM` [↗](#) (page 115) to persist files across sessions, we recommend that you use the `SASUSER` [↗](#) (page 107) or `USER` [↗](#) (page 111) system options to create a location in which you can store permanent files.

Example

In this example, the system option is specified on the command line:

```
WPS d:\temp\out1.wps -WORK d:\tempwork
```

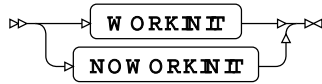
Any datasets created by the program `out1.wps` are written to the folder `d:\tempwork`. These can be used while the program is running, but when the session ends, the datasets are deleted.

To retain the datasets, also specify `NOWORKTERM`:

```
WPS d:\temp\out1.wps -WORK d:\tempwork -NOWORKTERM
```

WORKINIT

Specifies whether files in the `WORK` library location are deleted when WPS starts.



Valid in:	Configuration file and command line.
Default:	NOWORKINIT
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	False

If this system option is specified, all pre-existing content in the `WORK` library location is deleted when the WPS session starts.

WORKINIT

Delete files.

NOWORKINIT

Do not delete files.

Although `NOWORKINIT` and `NOWORKTERM` [↗](#) (page 115) can be used to persist datasets across sessions, we recommend that you use the `SASUSER` [↗](#) (page 107) or `USER` [↗](#) (page 111) system options to create a location in which you can store files you want to keep.

Example

In this example, the system option is specified on the command line. First, a program is run with the `NOWORKTERM` system option so that the datasets are retained in the working location.

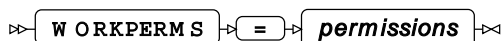
```
WPS c:\temp\out1.wps -NOWORKTERM
```

Then, the following program is run with the system option specified, so that the session retains the datasets in the working location:

```
WPS c:\temp\in1.wps -NOWORKINIT
```

WORKPERMS

Specifies the permission level of the `WORK` library location.



Valid in:	Configuration file and command line.
Default:	700
Maximum length:	3
Option group:	ENVFILES
Portable	False
Restrictable	False
Saveable	False
Supported platform:	AIX for pSeries 64-bit Linux for ARM Linux for pSeries Linux (LE) for pSeries Linux for System z 64-bit Linux for System z 64-bit Linux 32-bit Linux 64-bit Mac O/S Solararis for SPARC Solaris for 64-bit x86 Solaris for 32-bit x86 z/OS for System z

permissions

The permission level. This is expressed using Unix-style octal numeric notation for file system permissions.

For example, the default setting `700` specifies that the owner can read from, write to, or execute files in directories the `WORK` location, but no other user can access the location.

When a WPS session starts a `WORK` location is created. For example, on Linux, by default the `WORK` library location is created as a subdirectory of `/tmp`. Permissions are set for the `WORK` location, not for the directory that contains it.

Example

In this example, the system option is specified on the command line on Linux. A `WORK` library location is specified and the permissions set for any directories created in that location.

```
/opt/wps/bin/wps tscr.wps -work 'temp' -noworkterm -workperms 700
```

This created a subdirectory `temp` in the current directory, which is used as the current `WORK` library location. The system option `NOWORKTERM` was also specified so that the working lo

The `ls` command is then used to display the permissions of the directory:

```
ls -l temp
```


This displays the listing for the `WORK` location in `temp`:

```
drwx-----. 2 sd sd 21 Oct  5 10:33 WPS_work0547000022da_ian-bd-vm2.teamwpc.local
```

Only the owner has access to the directory.

WORKTERM

Specifies whether files in the `WORK` library location are deleted when the WPS session finishes.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOWORKTERM
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	True

If this system option is specified, all content in the `WORK` library location is deleted when the WPS session ends.

WORKTERM

Delete files.

NOWORKTERM

Do not delete files.

Although `NOWORKINIT` [↗](#) (page 113) and `NOWORKTERM` can be used to persist datasets across sessions, we recommend that you use the `SASUSER` [↗](#) (page 107) or `USER` [↗](#) (page 111) system options to create a location in which you can store files you want to keep.

Example

In this example, the system option is specified on the command line. First, a program is run with the `NOWORKTERM` system option so that the datasets are retained in the working location.

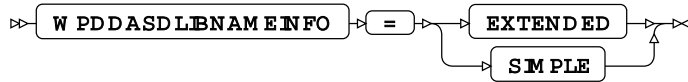
```
WPS c:\temp\out1.wps -NOWORKTERM
```

Then, the following program is run with the system option specified, so that the session retains the datasets in the working location:

```
WPS c:\temp\in1.wps -NOWORKINIT
```

WPDDASDLIBNAMEINFO

Specifies that extra information is returned for WPD DASD libraries.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	SIMPLE
Option group:	ENVFILES
Portable	True
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

This option enables additional information to be reported about DASD libraries:

- The maximum number of blocks that have been used by container.
- The number of blocks that are currently used.
- The number of blocks that are currently free.

EXTENDED

Provide the extra information.

SIMPLE

Do not provide the extra information.

Note:

Specifying this option changes how `DICTIONARY.LIBNAME`, `SASHELP.VLIBNM` and `LIBNAME LIST` describe a library.

Example

In this example, the `OPTIONS` statement is used to specifies that extra information is returned for WPD DASD libraries.

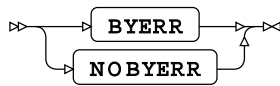
```
OPTIONS WPDDASDLIBNAMEINFO = EXTENDED;
```

ERRORHANDLING group system options

The system options in this group specify settings that affect error handling.

BYERR

Specifies that an error is generated if a null dataset is used as input to the `SORT` procedure and there is no `BY` variable.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOBYERR`

Option group: `ERRORHANDLING`

Portable: `True`

Restrictable: `True`

Saveable: `True`

BYERR

Generate an error.

NOBYERR

Do not generate an error.

This option controls what happens when a `_NULL_` dataset is used with `PROC SORT` and no `BY` variables are specified.

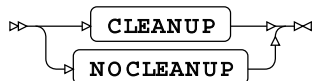
Example

In this example, the `OPTIONS` statement is used to specify that no error is generated if a null dataset and no `BY` variables are specified to `PROC SORT`.

```
OPTIONS NOBYERR;  
PROC SORT DATA=WORK._NULL_;  
BY;
```

CLEANUP

This system option is provided for compatibility only, and has no effect in WPS.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOCLEANUP`

Option group: `ERRORHANDLING`

Portable	True
Restrictable	True
Saveable	True

CLEANUP

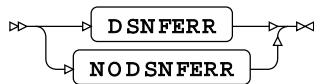
For compatibility only.

NOCLEANUP

For compatibility only.

DSNFERR

Specifies whether an error is generated when a specified dataset does not exist.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NODSNFERR
Option group:	ERRORHANDLING
Portable	True
Restrictable	True
Saveable	True

DSNFERR

An error is generated.

NODSNFERR

No error is generated and the program continues to run.

Example

In this example, the `OPTIONS` statement is used to specify that no error is generated if a specified dataset does not exist.

```
OPTIONS NODSNFERR;
LIBNAME books 'c:\temp\books';
DATA out;
  SET books.books_new;
  OUTPUT;
RUN;
```

In this example, the dataset `books_new` does not exist. The program runs without error.

If the system option is set:

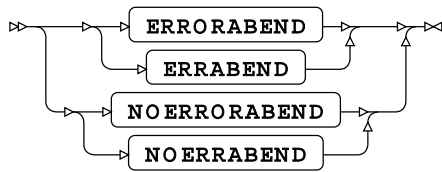
```
OPTIONS NODSNFERR
```

the program stops and an error message is written to the log:

```
ERROR: Data set "BOOKS.books_new" not found
```

ERRORABEND

Specifies whether a program stops running when an error occurs.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOERRORABEND`

Option group: `ERRORHANDLING`

Portable: `True`

Restrictable: `True`

Saveable: `True`

ERRORABEND

The program stops.

NOERRORABEND

The program continues. A note is written to the log.

Example

In this example, the `OPTIONS` statement is used to specify that a program continues to run if an error occurs.

```
OPTIONS NOERRORABEND;  
DATA _NULL_;  
  PUR "The program starts here";  
RUN;  
  
DATA _NULL_;  
  PUT "The program continues to here";  
RUN;
```

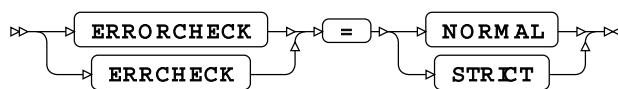
Because `NOERRORABEND` is set, the error in the first `DATA` step is noted, but the program continues to run through the second `DATA` step.

If **ERRORABEND** had been specified instead, the program would have stopped at the error in the first **DATA** step, with the message:

```
ERROR: The statement "pur" is unknown in this context
ERROR: Processing of the program will terminate because an error has occurred and
the ERRORABEND
      system option is in effect
```

ERRORCHECK

Specifies what happens if a file specified to **%INCLUDE** is missing.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NORMAL
Maximum length:	10
Option group:	ERRORHANDLING
Portable	True
Restrictable	True
Saveable	True

NORMAL

If the specified file does not exist, an error message is written to the log, and the program stops.
The **SYSCC** automatic macro variable is set to 0.

STRICT

If the specified file does not exist, an error message is written to the log, and the program stops.
The **SYSCC** automatic macro variable is set to 3000.

Example

In this example, the **OPTIONS** statement is used to specify that the **SYSCC** is set to 0 if an included file does not exist.

```
FILENAME test 'c:\temp\test.wps';
OPTIONS ERRORCHECK=STRICT;
DATA _NULL_;
  %INCLUDE test1;
RUN;
%LET x = &SYSCC;
%PUT Result code is: &x;
```

This produces the following output:

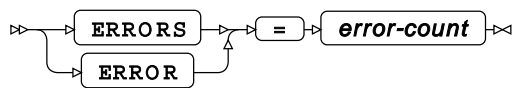
```
Result code is: 3000
```

If `ERRORCHECK` had been set to `NORMAL`, then the result is:

```
Result code is: 0
```

ERRORS

Specifies the maximum number of observations for which error messages are output.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	20
Minimum value:	0
Maximum value:	2147483647
Option group:	ERRORHANDLING LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

error-count

The number of observations for which error messages are output.

Example

In this example, the `OPTIONS` statement is used to specify that the program stops producing messages if the number of errors exceeds five.

```
OPTIONS ERRORS=5;
LIBNAME books XLSX 'c:\temp\books\books.xlsx';
DATA out;
    SET books.books1;
    IF title EQ 6 THEN OUTPUT;
RUN;
```

The `IF` statement attempts to compare a character value to a numeric value, which causes an error. This error is reported in the log for each observation, until the fifth is reached:

```
WARNING: Limit set by ERRORS= reached : No further messages of this type will be
printed
_N_=5 _ERROR_=1 _IORC_=0 Title=English Common Reader, The Type=History
Author=Altick, Richard D
Read=a Date_Read= Owned=n Y_Total=.
```

No more errors are reported after this observation.

FMterr

Specifies whether to treat missing user-defined formats as an error.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFMterr`

Option group: `ERRORHANDLING`

Portable: `True`

Restrictable: `True`

Saveable: `True`

FMterr

Missing formats are treated as an error. The program stops running and an error message is written to the log.

NOFMterr

Missing formats are not treated as an error. The program continues running. Values to which formats would have been applied remain unformatted.

Example

In this example, the `OPTIONS` statement is used to specify that no error is generated a user format is not found. The program creates user formats, but the `DATA` step has a mistyped format name.

```
PROC FORMAT;  
VALUE txt  
1='Zebra'  
2='Ocelot'  
3='Horse';  
  
OPTIONS NOFMterr;  
DATA _NULL_;  
  x = 1;  
  FORMAT x tct.;  
  PUT 'The animal is: ' x;  
RUN;
```

This produces the following output:

```
The animal is: 1
```

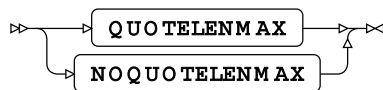
The program has run to the end, but the variable has not been formatted.

If `FMterr` is set, an error message is written to the log. In this example, the error message is:

```
ERROR: Format tct was not found or could not be loaded
```

QUOTELENMAX

Specifies whether to write a warning to the log output when quoted string literals exceed 262 characters.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOQUOTELENMAX`

Option group: `ERRORHANDLING`

Portable: `True`

Restrictable: `True`

Saveable: `True`

QUOTELENMAX

Print a warning.

NOQUOTELENMAX

Do not print a warning.

Example

In this example, the `OPTIONS` statement is used to specify that a warning is written if quoted string literals exceed 262 characters.

```
OPTIONS QUOTELENMAX;
DATA _NULL_;
    z = 'The Magnificent Cycle Company of Dyreham, 1 Forgotten Lane, Hampstead,
    Dyreham,
    The Magnificent Car Company of Wessex, 2 Forgotten Lane, Hampstead, Dyreham,
    The Esteemed West Country Steel Company, 6 Forgotten Lane, Hampstead, Dyreham,
    The Local Cheese Company, 10 Forgotten Lane, Hampstead, Dyreham,
    Bread and Cakes, 100 Southleigh Road, Dyreham';
    ll = length(z);
    put 'Length of string is: ' ll;

RUN;
```

The program runs to the end, and writes the following to the log:

```
Length of string is: 342
The Magnificent Cycle Company of Dyreham, 1 Forgotten Lane, Hampstead, Dyreham,The
  Magnificent C
ar Company of Wessex, 2 Forgotten Lane, Hampstead, Dyreham,The Esteemed West Country
  Steel Compa
ny, 6 Forgotten Lane, Hampstead, Dyreham,The Local Cheese Company, 10 Forgotten
  Lane, Hampstead,
  Dyreham,Bread and Cakes, 100 Southleigh Road, Dyreham
```

However, a warning is also written:

```
WARNING: A quoted string literal has become more than 262 characters in length.
  The literal currently contains: 'The Magnificent Cycle Company of Dyreham,
  1 Forgotten
    Lane, Hampstead, Dyreham,The Magnificent Car Company of Wessex, 2 Forgotten
  Lane,
    Hampstead, Dyreham,The Esteemed West Country Steel Company, 6 Forgotten
  Lane,
    Hampstead, Dyreham,The Local Cheese Company, 10 F'
```

VNFERR

Specifies whether to generate an error if there are missing variables.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NOVNFERR
Option group:	ERRORHANDLING
Portable	True

Restrictable True

Saveable True

VNFERR

An error is generated.

NOVNFERR

An error is not generated.

Example

In this example, the `OPTIONS` statement is used to specify that an error is not generated if there are missing variables.

```
DATA short1;
  v1 = 'abc';
  v2 = 1;
  v3=1;
  v4=1;
  OUTPUT;
RUN;

OPTIONS NOVNFERR;
DATA out;
  MERGE short1 _null_;
  BY v1 v2 v3;
  PUT 'Has run to end';
RUN;
```

In this example, the first `DATA` step creates dataset with four variables. The second `DATA` step attempts to merge the created dataset with a `_NULL_` dataset. The `_NULL_` dataset does not contain the variables contained in the `BY` statement. The second `DATA` step writes `Has run to end` to the log. However, warning messages are also written:

```
WARNING: BY variable "v1" does not exist in the data set "WORK._null_"
WARNING: BY variable "v2" does not exist in the data set "WORK._null_"
WARNING: BY variable "v3" does not exist in the data set "WORK._null_"
```

If the system option had been set to `NOVNFERR`, the second `DATA` step would have stopped with errors:

```
ERROR: BY variable "v1" does not exist in the data set "WORK._null_"
ERROR: BY variable "v2" does not exist in the data set "WORK._null_"
ERROR: BY variable "v3" does not exist in the data set "WORK._null_"
```

EXECMODES group system options

The system options in this group specify environment settings for files.

CONFIGFONT

Specifies whether to generate the font configuration installation file.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOCONFIGFONT</code>
Option group:	<code>EXECMODES</code>
Portable	True
Restrictable	True
Saveable	True
Supported platform:	AIX for pSeries 64-bit Linux for ARM Linux for pSeries Linux (LE) for pSeries Linux for System z 64-bit Linux for System z 64-bit Linux 32-bit Linux 64-bit Mac O/S Solararis for SPARC Solaris for 64-bit x86 Solaris for 32-bit x86 z/OS for System z

CONFIGFONT

Do not generate the font configuration installation file.

NOCONFIGFONT

Do not generate the font configuration installation file.

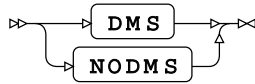
Example

In this example, the `OPTIONS` statement is used to specify that the font configuration file is not generated.

```
OPTIONS NOCONFIGFONT;
```

DMS

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	Configuration file and command line.
Default:	NODMS
Option group:	EXECMODES
Portable	False
Restrictable	True
Saveable	False

DMS

For compatibility only.

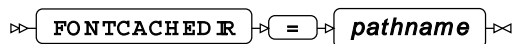
NODMS

For compatibility only.

If you specify `DMS`, an error message is displayed.

FontCACHEDIR

Specifies the location of the WPS font cache directory, used by ODS to cache fonts.



Valid in:	Configuration file and command line.
Maximum length:	32767
Option group:	INSTALL
Portable	False
Restrictable	True
Saveable	True

The specified directory is used to cache fonts required for ODS PDF fonts.

pathname

The path to the location.

This location is only used if necessary to cache the fonts. Some operating systems automatically set up cache when required, and this is used as a backup location.

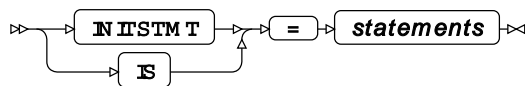
Example

In this example, the system option is specified on the command line:

```
WPS c:\temp\print.wps -FONTCACHEDIR c:\temp\fontcache
```

INITSTMT

Specifies statements to execute before any submitted SAS language program.



Valid in:	Configuration file and command line.
Maximum length:	2048
Option group:	EXECMODES
Portable	True
Restrictable	False
Saveable	False

statements

The list of statements to execute before a submitted program.

Statements can include procedures, data steps, global statements, and so on. The statements must be enclosed in double quotations marks (").

Example

In this example, the system option is specified on the command line, and specifies DATA step statements that are to be run before the submitted DATA step.

```
wps c:\temp\test.wps -initstmt "DATA _NULL_;PUT 'The initial statements output this message';RUN;"
```

If c:\temp\test.wps contains:

```
DATA _NULL_;
  PUT 'This program runs after statements in INITSTMT have been processed';
RUN;
```

then the following is written to the log:

```
The initial statements output this message
NOTE: The data step took :
      real time : 0.014
      cpu time  : 0.000

1
2      DATA _NULL_;
3          PUT 'These statements run after statements in INITSTMT have been
processed';
4      RUN;

These statements run after statements in INITSTMT have been processed
NOTE: The data step took :
      real time : 0.044
      cpu time  : 0.000

NOTE: Submitted statements took :
      real time : 0.273
      cpu time  : 0.125
```

LINKINITSTMT

Specify statements to execute on the local server before any program submitted to a WPS Link server.

```
>> LINKINITSTMT = << statements ><
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	2048
Option group:	EXECMODES
Portable	True
Restrictable	True
Saveable	False

statements

The statements to be executed. These must be enclosed in quotation marks.

statements can contain any SAS language statements. You can use this system option to execute statements that write information to the log such as user identifiers and the time at which the program was submitted; for example:

```
%put Current user: &SYSUSERID;%SYSFUNC(datetime(),datetime22.3);
```

This system option can be useful for recording user activity with the WPS Workbench, providing information for compliance purposes.

If the system option is used for compliance or security purposes, we recommend that the system options are restricted, so that users cannot change their values. For information on restricting system options, see *Restricting system options* [↗](#) (page 46).

You can also execute statements when the submit to WPS Link has finished; see `LINKTERMSTMT` [↗](#) (page 130).

Example

In this example, the `OPTIONS` statement is used to specify statements to execute before any submitted program to a WPS Link server.

```
OPTIONS LINKINITSTMT= "%put The user: &SYSUSERID ran this job with
process ID: &SYSPROCESSID on: %SYSFUNC(datetime(),datetime22.3);";
```

The following is written to the log when a WPS Link program is submitted:

```
Current user: sd ran this job with process ID: 41DBD1EBA430F5C30000000000000000 on:
28FEB2019:16:21:08.689
```

LINKTERMSTMT

Specify statements to execute on the local server after any program submitted to a WPS Link server.

```
LINKTERMSTMT = statements
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	2048
Option group:	<code>EXECMODES</code>
Portable	True
Restrictable	True
Saveable	False

statements

The statements to be executed. These must be enclosed in quotation marks.

statements can contain any SAS language statements. You can use this system option to execute statements that write information to the log such as user identifiers and the time at which the program finished executing; for example:

```
%put Current user: &SYSUSERID;%SYSFUNC(datetime(),datetime22.3);
```


This system option can be useful for recording user activity with the WPS Workbench, providing information for compliance purposes.

If the system option is used for compliance or security purposes, we recommend that the system options are restricted, so that users cannot change their values. For information on restricting system options, see *Restricting system options* [↗](#) (page 46).

You can also execute statements when the submit to WPS Link has finished; see `LINKTERMSTMT` [↗](#) (page 130).

Example

In this example, the `OPTIONS` statement is used to specify statements to execute after a program submitted to a WPS Link server has executed.

```
OPTIONS LINKTERMSTMT= "%put The user: &SYSUSERID ran this job with
process ID: &SYSPROCESSID on: %SYSFUNC(datetime(),datetime22.3);";
```

The following is written to the log after the submitted program has finished executing:

```
Current user: sd ran this job with process ID: 41DBD1EBA430F5C30000000000000000 on:
28FEB2019:16:22:10.001
```

JREOPTIONS

Specifies options for the Java Runtime Environment (JRE).

```
⟨JREOPTIONS⟩ = ⟨(⟩ ⟨runtime-options⟩ ⟨)⟩
```

Valid in:	Configuration file and command line.
Default:	<empty-string>
Maximum length:	32000
Option group:	EXECMODES
Appendable	True
Portable	False
Restrictable	True
Saveable	False

runtime-options

The options, in parentheses, for the JRE. For example, when set in a configuration file:

```
-JREOPTIONS ('-Djava.class.path=!wpshome\jars\wpcnet.jar;!wpshome\jars
\wpcjocl.jar;!wpshome\jars\wpsssh.jar')
```

Example

In this example, the option is set in a configuration file:

```
-JREOPTIONS ('-Djava.class.path=!wpshome\jars\wpcnet.jar;!wpshome\jars\wpcjocl.jar;!wpshome\jars\wpsssh.jar')
```

SCANDEFAULTMODIFIERS

Specifies the default modifiers for the `SCAN DATA` step function.



Valid in:	Configuration file and command line.
Maximum length:	1024
Option group:	EXECSMODES
Portable	True
Restrictable	True
Saveable	True

The `SCAN DATA` step function returns the word at a specified position in a string. Various modifiers can be set that specify how the string is searched, what characters are viewed as delimiters, what type of characters are considered, and so on. The modifiers are set to default values by WPS, but you can set the defaults yourself using this system option.

modifiers

If more than one modifier is specified, specify them as a string without separators; for example, `BMOQ`. The modifiers are the same as you would use in the `DATA` step function. `SCAN`

Example

In this example, the system option is specified on the command line.

```
wps -scandefaultmodifiers HM c:\temp\program1.wps
```

This sets the default modifiers for `SCAN` to `H` and `M`. The program `program1.wps` contains the `DATA` step:

```
DATA _NULL_;
  result1 = SCAN('london,,,bike,company;;A1', 3, ',;');
  PUT result1=;
RUN;
```

By default, `SCAN` treats multiple adjacent separators as one separator. If you set `M`, adjacent separators are treated separately. In this example, `SCAN` returns null as `M` has been set using `SCANDEFAULTMODIFIERS`.

TERMSTMT

Specifies statements to execute after any submitted SAS language program.

➤ **TERMSTMT** ➤ = ➤ **statements** ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	2048
Option group:	EXECMODES
Portable	True
Restrictable	True
Saveable	False

statements

The list of statements to execute after the submitted program.

Statements can include procedures, data steps, global statements, and so on. The statements must be enclosed in double quotations marks (").

Example

In this example, the system option is specified on the command line, and specifies DATA step statements that will execute after the submitted DATA step.

```
wps c:\temp\test.wps -termstmt "DATA _NULL_;PUT 'The terminal statements output this message';RUN;"
```

The file `c:\temp\test.wps` contains:

```
DATA _NULL_;  
  PUT 'These statements run before statements in TERMSTMT have been processed';  
RUN;
```

The following is written to the log:

```
1      DATA _NULL_;  
2          PUT 'These statements run before statements in TERMSTMT have been  
processed';  
3      RUN;
```

These statements run before statements in TERMSTMT have been processed

NOTE: The data step took :
 real time : 0.041
 cpu time : 0.000

The terminal statements output this message

NOTE: The data step took :
 real time : 0.017
 cpu time : 0.000

NOTE: Submitted statements took :
 real time : 0.279
 cpu time : 0.171

EXTFILES group system options

The system options in this group specify settings for external files.

BOMFILE

Specifies whether a Byte Order Mark (BOM) prefix is written to external files that have Unicode encoding.



Valid in: OPTIONS statement, configuration file and command line.

Default: NOBOMFILE

Option group: EXTFILES

Portable True

Restrictable True

Saveable True

BOMFILE

Write a BOM prefix.

NOBOMFILE

Do not write a BOM prefix.

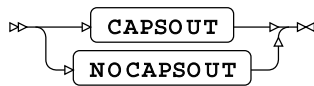
Example

In this example, the `OPTIONS` statement is used to specify that no byte order mark is written.

```
OPTIONS NOBOMFILE;
```

CAPSOUT

Specifies whether listing output written to an MVS file is converted to upper case.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOCAPSOUT`

Option group: `EXTFILES`

Portable: `True`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

CAPSOUT

Convert output to upper case.

NOCAPSOUT

Do not convert output to upper case.

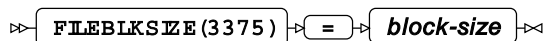
Example

In this example, the `OPTIONS` statement is used to specify that output is not converted to upper case.

```
OPTIONS CAPSOUT;
```

FILEBLKSIZE(3375)

Specifies the default block size for data libraries on 3375 devices.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	17600
Minimum value:	5
Maximum value:	32760
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the OPTIONS statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(3375) = 15552
```

FILEBLKSIZE(3380)

Specifies the default block size for data libraries on 3380 devices.

```
FILEBLKSIZE(3380) = block-size
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	23476
Minimum value:	5
Maximum value:	32760
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(3375) = 17600;
```

FILEBLKSIZE(3390)

Specifies the default block size for data libraries on 3390 devices.

⇒ **FILEBLKSIZE(3390)** ⇒ = ⇒ **block-size** ⇒

Valid in:	OPTIONS statement, configuration file and command line.
Default:	27998
Minimum value:	5
Maximum value:	32760
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that connections continue to exist after `ENDRSUBMIT` statements.

```
OPTIONS FILEBLKSIZE(3390) = 15552;
```

FILEBLKSIZE(3400)

Specifies the default block size for data libraries on 3400 devices.

⇒ **FILEBLKSIZE(3400)** ⇒ = ⇒ **block-size** ⇒

Valid in:	OPTIONS statement, configuration file and command line.
Default:	32760

Minimum value:	5
Maximum value:	32760
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(3400) = 15552;
```

FILEBLKSIZE(3480)

Specifies the default block size for data libraries on 3480 devices.

➤ **FILEBLKSIZE(3480)** ➤ = ➤ ***block-size*** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	32760
Minimum value:	5
Maximum value:	65535
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that connections continue to exist after `ENDRSUBMIT` statements.

```
OPTIONS FILEBLKSIZE(3480) = 27998;
```

FILEBLKSIZE(3490E)

Specifies the default block size for data libraries on 3490E devices.

➤ **FILEBLKSIZE(3490E)** ➤ = ➤ **block-size** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	32760
Minimum value:	5
Maximum value:	65535
Option group:	<code>EXTFILES</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(3490E) = 17600;
```

FILEBLKSIZE(3590)

Specifies the default block size for data libraries on 3590 devices.

➤ **FILEBLKSIZE(3590)** ➤ = ➤ **block-size** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	32760

Minimum value:	5
Maximum value:	262144
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(3590) = 27998;
```

FILEBLKSIZE(9345)

Specifies the default block size for data libraries on 9345 devices.

➤ **FILEBLKSIZE(9345)** ➤ **=** ➤ ***block-size*** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	22928
Minimum value:	5
Maximum value:	32760
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

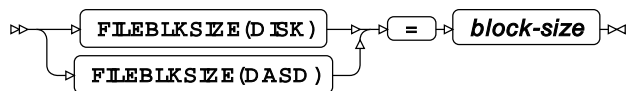
Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(9345) = 15552;
```

FILEBLKSIZE(DISK)

Specifies the default block size for external files on `DISK` devices.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	8
Option group:	<code>EXTFILES</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

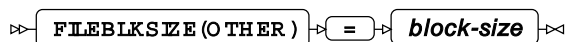
Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(DISK) = 15552;
```

FILEBLKSIZE(OTHER)

Specifies the default block size for external files on `OTHER` devices.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	6400
Minimum value:	5
Maximum value:	6400

Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(OTHER) = 32000;
```

FILEBLKSIZE(SYSOUT)

Specifies the default block size for external files on `SYSOUT` devices.

➤ **FILEBLKSIZE(SYSOUT)** ➤ **=** ➤ ***block-size*** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	264
Minimum value:	5
Maximum value:	264
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(SYSOUT) = 128;
```

FILEBLKSIZE(TAPE)

Specifies the default block size for external files on `TAPE` devices.

⇒ **FILEBLKSIZE(TAPE)** ⇒ = ⇒ *block-size* ⇒

Valid in: `OPTIONS` statement, configuration file and command line.
Maximum length: 8
Option group: `EXTFILES`
Supported platform: z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(TAPE) = 256;
```

FILEBLKSIZE(TERM)

Specifies the default block size for external files on `TERM` devices.

⇒ **FILEBLKSIZE(TERM)** ⇒ = ⇒ *block-size* ⇒

Valid in: `OPTIONS` statement, configuration file and command line.
Default: 264
Minimum value: 5
Maximum value: 264
Option group: `EXTFILES`
Supported platform: z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

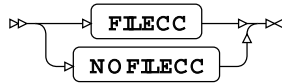
Example

In this example, the `OPTIONS` statement is used to specify the block size.

```
OPTIONS FILEBLKSIZE(TERM) = 256;
```

FILECC

Specifies whether external datasets are checked for the `PRINT` attribute.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFILECC`

Option group: `EXTFILES`

Supported platform: `z/OS for System z`

FILECC

Check datasets.

NOFILECC

Do not check datasets.

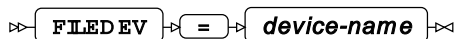
Example

In this example, the `OPTIONS` statement is used to specify that external datasets are not checked for the `PRINT` attribute.

```
OPTIONS NOFILECC;
```

FILEDEV

Specifies the default device to be used for new physical files on `z/OS`.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `SYSDA`

Maximum length: `8`

Option group: `EXTFILES`
`SASFILES`

Supported platform: `z/OS for System z`

device-name

The name of the default device. This can be up to eight characters long.

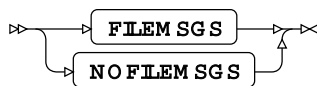
Example

In this example, the `OPTIONS` statement is used to specify that the default device is `VIO`.

```
OPTIONS FILEDEV=VIO;
```

FILEMSGS

Specifies whether to write messages to the log about the results of dynamic allocations using `DDNAME`.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFILEMSGS`

Option group: `EXTFILES`
`SASFILES`

Supported platform: `z/OS for System z`

FILEMSGS

Write messages to the log.

NOFILEMSGS

Do not write messages to the log.

A message written to the log during dynamic allocation has the following form:

```
1 message from dynalloc
1GD103I SMS ALLOCATED TO DDNAME CAT SYS0008
```

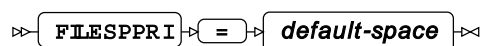
Example

In this example, the `OPTIONS` statement is used to specify that dynamic allocation log messages are not display.

```
OPTIONS NOFILEMSGS;
```

FILESPPRI

Specifies the default primary space allocation for new physical files.



Valid in: `OPTIONS` statement, configuration file and command line.

Default:	1
Minimum value:	1
Maximum value:	32760
Option group:	EXTFILES SASFILES
Supported platform:	z/OS for System z

default-space

The default primary space allocation.

The space is allocated in the units specified by the system option `FILEUNIT` [↗](#) (page 147).

Example

In these examples, the `OPTIONS` statement is used to specify the size of the primary space allocation.

```
OPTIONS FILEUNIT = CYL FILESPPRI = 2;
```

This sets the default primary space allocation to two cylinders.

```
OPTIONS FILEUNIT = 1024 FILESPPRI = 56;
```

This sets the default primary space allocation to 56 units of 1024 bytes, or 57,344 bytes.

FILESPSEC

Specifies the default secondary space allocation for new physical files.

➤ **FILESPSEC** ➤ = ➤ ***default-space*** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	0
Maximum value:	32760
Option group:	EXTFILES SASFILES
Supported platform:	z/OS for System z

default-space

The default secondary space allocation.

The space is allocated in the units specified by the system option `FILEUNIT` [↗](#) (page 147).

Example

In these examples, the `OPTIONS` statement is used to specify the size of the secondary space allocation.

```
OPTIONS FILEUNIT = CYL FILESPPRI = 1;
```

This sets the default secondary space allocation to one cylinders.

```
OPTIONS FILEUNIT = 1024 FILESPPRI = 56;
```

This sets the default secondary space allocation to 57,344 bytes.

FILESTAT

Specifies whether to maintain ISPF member statistics in partitioned datasets (PDS) or extended partitioned datasets (PDSE).



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFILESTAT`

Option group: `EXTFILES`

Supported platform: `z/OS for System z`

FILESTAT

Maintain ISPF member statistics in partitioned data sets.

NOFILESTAT

Do not maintain ISPF member statistics in partitioned data sets.

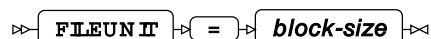
Example

In this example, the `OPTIONS` statement is used to specify that ISPF member statistics are not maintained in partitioned data sets.

```
OPTIONS NOFILESTAT;
```

FILEUNIT

Specifies the default unit of allocation for new physical files.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	CYL
Maximum length:	8
Option group:	EXTFILES SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The default block-size. The size can be specified as a number of bytes, or as a string. The string can be:

BLKS or BLK	Blocks
CYLS or CYL	Cylinders
TRKS or TRK	Tracks

This option enables you to specify the default unit of allocation as either a number of bytes or as an element of disk storage, such as a cylinder or track. These units can then be specified to other system options, such as `FILESPPRI` and `FILESPSEX`. For example, you might set the default unit of allocation as blocks. You can then use this as the unit to when specifying `FILESPPRI`. Setting `FILESPPRI = 16` would then set the primary space allocation for new physical files to 16 blocks. The number of bytes that this specifies depends on the type of device to which the allocation applies. If you specify the default unit as a number of bytes, that number of bytes is used as the unit. For example, you might set the default unit of allocation as 1024 bytes; specifying `FILESPPRI = 24` would then set the primary space allocation for new physical files to 24576 bytes.

Example

In this example, the `OPTIONS` statement is used to specify the default unit of allocation.

```
OPTIONS FILEUNIT = TRK;
```

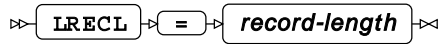
The default unit of allocation is tracks.

```
OPTIONS FILEUNIT = 2048;
```

The default unit of allocation is 2048 bytes.

LRECL

Specifies the default record length for input and output files.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	256
Minimum value:	1
Maximum value:	32767
Option group:	EXTFILES
Portable	True
Restrictable	True
Saveable	True

record-length

The default record length.

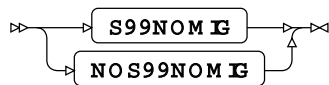
Example

In this example, the `OPTIONS` statement is used to specify the default record length.

```
OPTIONS LRECL=320;
```

S99NOMIG

Specifies whether to restore migrated datasets.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOS99NOMIG
Option group:	EXTFILES SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

S99NOMIG

Restore migrated data sets.

NOS99NOMIG

Do not restore migrated data sets.

Example

In this example, the `OPTIONS` statement is used to specify that migrated datasets are not restored.

```
OPTIONS NOS99NOMIG;
```

SYSPREF

Specifies a prefix for partially-qualified physical file names.

SYSPREF = *prefix*

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	42
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

prefix

The prefix for the high-level qualifier. By default, the host system's default high-level qualifier is used.

SYSTEMLOCALEENCODING

Specifies that filenames created in a WPS session are transcoded using the device's locale.

SYSTEMLOCALEENCODING
NO SYSTEMLOCALEENCODING

Valid in:	Configuration file and command line.
-----------	--------------------------------------

Default:	NOSYSTEMLOCALEENCODING
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True

This system options enables a filename created using the session encoding to be transcoded to the system locale encoding when writing the file to a device. For example, if a WPS session using Latin1 session encoding creates a file on a device using UTF-8 encoding, the filename is transcoded correctly.

SYSTEMLOCALEENCODING

Filenames are transcoded to the device locale.

NOSYSTEMLOCALEENCODING

Filenames are not transcoded to the device locale.

This system options applies to WPS on Linux and UNIX, not z/OS, Windows or Macintosh.

Example

In this example, the `OPTIONS` statement is used to specify that filenames are transcoded.

```
OPTION SYSTEMLOCALEENCODING;
```

TRANSACTIONFILELOCKINGBLOCKS

Specifies what happens when a dataset cannot be opened for read or write because another process has locked it.



Valid in:	OPTIONS statement, configuration file and command line.
Option group:	EXTFILES
Portable	True
Restrictable	True
Saveable	True

A process cannot open a dataset or catalog for a read or write operation if another process already has access to the file. Use this option to specify what happens if the file cannot be locked.

TRANSACTIONALFILELOCKINGBLOCKS

The process will request a lock, and then wait for the current activity on the file to finish and for its own lock to be granted. Any other activity on the file by any other process is blocked.

NOTTRANSACTIONALFILELOCKINGBLOCKS

If a process cannot access the file, the operation fails. The process waits for the file to become available but times out with an error and terminates the current process.

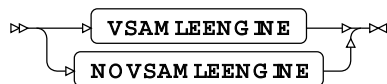
Example

In this example, the `OPTIONS` statement is used to specify that if a transacted file operation is blocked, the operation does not wait for the file to be released.

```
OPTION NOTTRANSACTIONALFILELOCKINGBLOCKS;
```

VSAMLEENGINE

Specifies whether to use the old or new VSAM engines in WPS.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOVSAMLEENGINE`

Option group: `EXTFILES`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

VSAMLEENGINE

Use the old VSAM engine.

NOVSAMLEENGINE

Use the new VASM engine.

The old VSAM engine uses the IBM Language Environment, whereas the new VSAM engine does not.

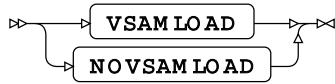
Example

In this example, the `OPTIONS` statement is used to specify that new VSAM engine is used.

```
OPTIONS NOVAMLEENGINE;
```

VSAMLOAD

Specifies whether empty VSAM datasets can be loaded.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOVSAMLOAD
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

VSAMLOAD

Empty VSAM datasets can be loaded.

NOVSAMLOAD

Empty VSAM datasets cannot be loaded.

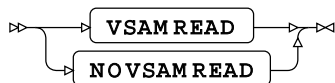
Example

In this example, the `OPTIONS` statement is used to specify that empty VSAM datasets cannot be loaded.

```
OPTIONS NOVSAMLOAD;
```

VSAMREAD

Specifies whether a VSAM dataset can be read using an `INFILE` statement.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOVSAMREAD
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True

Supported platform: z/OS for System z

VSAMREAD

A VSAM dataset can be read.

NOVSAMREAD

A VSAM dataset cannot be read.

Example

In this example, the `OPTIONS` statement is used to specify that a VSAM dataset cannot be read.

```
OPTIONS NOVSAMREAD;
```

VSAMRLS

Enables record-level sharing for VSAM datasets.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOVSAMRLS`

Option group: `EXTFILES`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: z/OS for System z

VSAMRLS

Record-level sharing is enabled.

NOVSAMRLS

Record-level sharing is disabled.

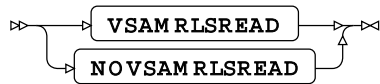
Example

In this example, the `OPTIONS` statement is used to disable record-level sharing for VSAM datasets.

```
OPTIONS NOVSAMRLS;
```


VSAMRLSREAD

Specifies whether the level of integrity for VSAM read operations can be set.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOVSAMRLSREAD
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

VSAMRLSREAD

Level of integrity can be set.

NOVSAMRLSREAD

Level of integrity cannot be set.

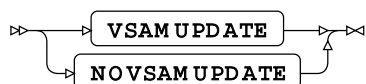
Example

In this example, the `OPTIONS` statement is used to specify that the VSAM read operations cannot be set.

```
OPTIONS VSAMRLSREAD;
```

VSAMUPDATE

Specifies whether a VSAM dataset opened using an `INFILE` statement can be updated.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOVSAMUPDATE
Option group:	EXTFILES
Portable	False
Restrictable	True
Saveable	True

Supported platform: z/OS for System z

VSAMUPDATE

A VSAM dataset can be updated.

NOVSAMUPDATE

A VSAM dataset cannot be updated.

Example

In this example, the `OPTIONS` statement is used to specify that VSAM datasets opened with the `INFILE` statement cannot be updated.

```
OPTIONS NOVSAMUPDATE;
```

FLE_CONTROL group system options

Specifies system options that control how WPS handles Python and R.

PYTHONHOME

Specifies the location of the Python installation.

```
» PYTHONHOME = filepath «
```

Valid in: `OPTIONS` statement, configuration file and command line.

Maximum length: 1024

Option group: `FLE_CONTROL`
`INSTALL`

Portable: True

Restrictable: True

Saveable: True

filepath

The location of the Python installation.

The Python installation location contains all the resources required to run a Python program; specifying the location enables WPS to run Python programs using the `PYTHON` procedure. The location must contain the `python.exe` executable to enable Python to run.

Example

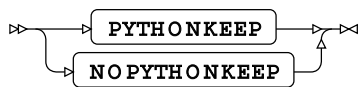
In this example, the option is specified on the command line.

```
wps c:\temp\sadd.wps -pythonhome "c:\program files\python"
```

The Python resources are found in the specified folder, and the program `sadd.wps` is executed.

PYTHONKEEP

Specifies whether to retain the Python environment after the Python procedure finishes.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOPYTHONKEEP
Option group:	FILE_CONTROL
Portable	True
Restrictable	True
Saveable	True

PYTHONKEEP

Retain the Python environment.

NOPYTHONKEEP

Do not retain the Python environment.

If **PYTHONKEEP** is specified, the Python environment is retained throughout the session, unless the **TERMINATE** option of a **PROC PYTHON** statement is specified, in which case the environment is terminated.

Example

In this example, the `OPTIONS` statement is used to specify that the Python environment is kept.

```
OPTIONS PYTHONKEEP;  
PROC PYTHON;  
SUBMIT;  
msg = "Hello World!"  
print (msg)  
ENDSUBMIT;  
RUN;  
  
PROC PYTHON;  
SUBMIT;  
print (msg)  
ENDSUBMIT;  
RUN;
```

The first program writes `Hello World!` to the location specified by the default ODS settings. Because the Python environment is retained, the second program also writes a message to the same location.

If `NOPYTHONKEEP` was set (the default), the Python environment is not retained, and the program fails. The following message is written to the log:

```
NOTE: NameError  
NOTE: :  
NOTE: name 'msg' is not defined
```

RKEEP

Specifies whether to retain the R environment after the R procedure finishes.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NORKEEP</code>
Option group:	<code>FILE_CONTROL</code>
Portable	True
Restrictable	True
Saveable	True

RKEEP

Retain the R environment.

NORKEEP

Do not retain the R environment.

If **RKEEP** is specified, the R environment is retained throughout the session, unless the **TERMINATE** option of a **PROC R** statement is specified, in which case the environment is terminated.

Example

In this example, the **OPTIONS** statement is used to specify that the R environment is not kept.

```
OPTIONS RKEEP;  
PROC R;  
SUBMIT;  
    myString <- "Hello, World!"  
    print ( myString)  
ENDSUBMIT;  
RUN;  
  
PROC R;  
SUBMIT;  
    print ( myString)  
ENDSUBMIT;  
RUN;
```

The first program writes `Hello World!` to the location specified by the default ODS settings. Because the T environment is retained, the second program also writes a message to the same location.

If **NORKEEP** is set (the default), the Python environment is not retained, and the program fails. The following message is written to the log:

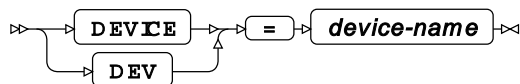
```
Error in print(myString) : object 'myString' not found
```

GRAPHICS group system options

Specifies system options that apply to graphics devices.

DEVICE

Specifies the device to be used for graphical output.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	8
Option group:	GRAPHICS
Portable	True
Restrictable	False

Saveable False

device-name

The name of the device to use. The device can be one of GIF, JPG, PNG, PDF, SVG, MULTI:PDF.

Example

In this example, the `OPTIONS` statement is used to specify that graphical output is written to PNG.

```
OPTIONS DEVICE = PNG;
```

MAPS

Specifies the location that contains map datasets.

➤ **MAPS** ➤ = ➤ *maps-source* ➤

Valid in:	Configuration file and command line.
Maximum length:	8192
Option group:	GRAPHICS
Appendable	True
Portable	True
Restrictable	True
Saveable	False

maps-source

The location.

By default, map datasets provided by WPS are stored in the installation location. For example, on Windows, the map datasets are stored in the folder *installation-folder*/maps, where *installation-folder* is the folder in which WPS was installed. This system option enables you to find map datasets in a specified location.

Example

In this example, the location of the map is specified on the command line.

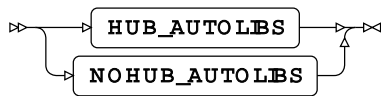
```
wps -maps 'c:\temp\maps' mapprog.wps
```

The GMAP procedure uses the mapping dataset required by `mapprog.wps` that is located in the folder `c:\temp\maps`.

HUB group system options

HUB_AUTOLIBS

Specifies whether all libraries specified in the Hub are automatically available for use by programs.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOHUB_AUTOLIBS
Option group:	HUB
Portable	True
Restrictable	True
Saveable	True

HUB_AUTOLIBS

Libraries are available.

NOHUB_AUTOLIBS

Libraries are not available.

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPB Hub, see the *WPS Hub User Guide*.

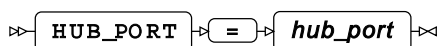
Example

In this example, the `OPTIONS` statement is used to specify that all libnames specified in the Hub are available to a program.

```
OPTIONS HUB_AUTOLIBS;
```

HUB_PORT

Specifies the port number of the WPS Hub server.



Valid in:	OPTIONS statement, configuration file and command line.
-----------	---

Default:	8080
Minimum value:	1
Maximum value:	65535
Option group:	HUB
Portable	True
Restrictable	True
Saveable	True

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPB Hub, see the *WPS Hub User Guide*.

hub_port

The port number.

Example

In this example, the `OPTIONS` statement is used to specify the WPS Hub port.

```
OPTIONS HUB_PORT = 8082;
```

HUB_PROTOCOL

Specifies whether to use HTTP or HTTPS to connect to the WPS Hub server.

⇒ `HUB_PROTOCOL` ⇒ = ⇒ `hub_protocol` ⇒

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	HTTPS
Option group:	HUB
Portable	True
Restrictable	True
Saveable	True

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPB Hub, see the *WPS Hub User Guide*.

hub_protocol

The protocol to use:

- HTTP
- HTTPS

Example

In this example, the `OPTIONS` statement is used to specify that HTTP is used to connect to WPS HUB.

```
OPTIONS HUB_PROTOCOL = HTTP;
```

HUB_PWD

Specifies the password required to log in to WPS Hub.

```
>> HUB_PWD = hub_pwd <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	80
Option group:	HUB
Appendable	False
Portable	True
Restrictable	True
Saveable	False

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPS Hub, see the *WPS Hub User Guide*.

hub_pwd

The password.

You can create an encoded password using the `PWENCODE DATA` step function, or the `PWENCODE` procedure.

Example

In this example, the `OPTIONS` statement is used to specify the password used to connect to WPS HUB.

```
OPTIONS HUB_PWD = HTTP;
```

HUB_SERVER

Specify the URL of the WPS Hub server.

```
>> HUB_SERVER = hub_server <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	255
Option group:	HUB
Appendable	False
Portable	True
Restrictable	True
Saveable	True

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPS Hub, see the *WPS Hub User Guide*.

hub_server

The URL.

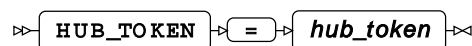
Example

In this example, the `OPTIONS` statement is used to specify the URL of WPS Hub.

```
OPTIONS HUB_SERVER = credentials-store;
```

HUB_TOKEN

Specifies a token to connect to the Web server.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	32767
Option group:	HUB
Appendable	False
Portable	True
Restrictable	True
Saveable	True

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPS Hub, see the *WPS Hub User Guide*.

hub_token

A token to be used as authorisation.

A token is generated using the hub.

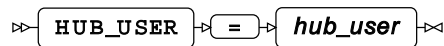
Example

In this example, the `OPTIONS` statement is used to specify that all libnames specified in the Hub are available to a program.

```
OPTIONS HUB_TOKEN = "xJ1yHb2zZ";
```

HUB_USER

Specifies the password required to log in to WPS Hub.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	80
Option group:	HUB
Appendable	False
Portable	True
Restrictable	True
Saveable	True

WPS Hub provides a central location for user credentials and resources that can be used with SAS language programs. For more information on WPB Hub, see the *WPS Hub User Guide*.

hub_user

The user name.

Example

In this example, the `OPTIONS` statement is used to specify the user name required to connect to WPS HUB.

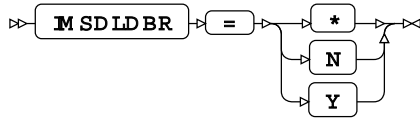
```
OPTIONS HUB_USER = "FredSmith";
```

IMS group system options

Specifies system options for the IMS.

IMSDLDBR

Specifies whether IMS sets the DBRC parameter when it invokes an IMS DLI region.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	Y
Option group:	IMS
Portable	False
Restrictable	False
Saveable	True
Supported platform:	z/OS for System z

Use the subsystem default.

N

The DBRC parameter is not set.

Y

The DBRC parameter is set.

Example

In this example, the `OPTIONS` statement is used to specify that the subsystem default is used.

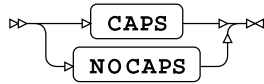
```
OPTIONS IMSDLDBR = *;
```

INPUTCONTROL group system options

System options that control input.

CAPS

Specifies whether strings specified in comparison operations are converted to upper case.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOCAPS
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

CAPS

Convert comparison strings to upper case.

NOCAPS

Do not convert comparison strings to upper case.

This system option can be useful if you want to run the same program with datasets that have similar variables, but where those values differ in case. By setting the system option appropriately, you can ensure that the same condition matches only the values in upper case. If the system option is set, any new strings added are also converted to upper case.

Example

In this example, the `OPTIONS` statement is used to specify that strings in the comparisons are treated as upper case.

```
OPTIONS CAPS;
DATA test;
  INPUT author $13. title $20.;
  if author EQ "Rendell, Ruth" then do;
    newob = CATX(' ', author, title, 'Novel');
    put newob ;
  end;
datalines;
Rendell, Ruth Going Wrong
ReNdElL, RUTH The Bridesmaid
REnDELL, RUTH The Tree of Hands
;
```

This produces the following output:

```
REnDELL, RUTH The Tree of Hands NOVEL
```

Only one of the observations is written to the log, the one in which the string in the *author* variable consists entirely of upper-case characters. Neither value containing lower-case letters matches the condition. The string `Novel`, which is concatenated to the other values, is also converted to upper-case characters.

CARDIMAGE

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOCARDIMAGE
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

CARDIMAGE

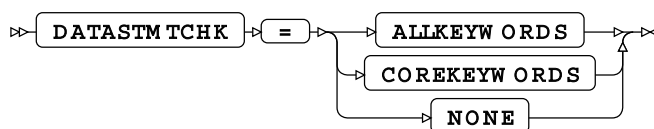
For compatibility only.

NOCARDIMAGE

For compatibility only.

DATASTMTCHK

Specifies whether dataset names can be the same as SAS language keywords.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	COREKEYWORDS
Option group:	INPUTCONTROL
Portable	True
Restrictable	True

Saveable True

ALLKEYWORDS

No WPS-recognised, SAS-language keyword can be used as a dataset name.

COREKEYWORDS

The keywords `MERGE`, `RETAIN`, `SET` and `UPDATE` cannot be used as dataset names.

NONE

Keywords can be used as dataset names.

By default, all SAS language keywords can be used as dataset names, except those associated with dataset and variable operations listed under `COREKEYWORDS`. You might, however, want to ensure that no dataset can be accidentally overwritten because keywords and dataset names are the same, in which case you can set `ALLKEYWORDS`.

Example

In this example, the `OPTIONS` statement is used to specify that no keywords can be used as dataset names.

```
OPTIONS DATASTMTCHK = ALLKEYWORDS;
```

If you were to then run the following `DATA` step:

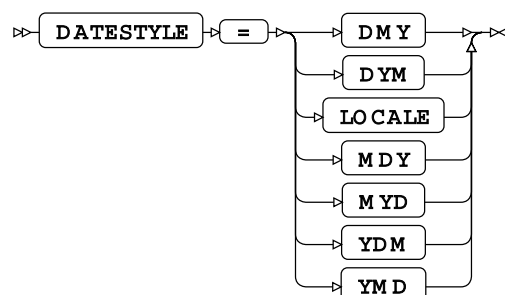
```
DATA WHERE;  
    x = 2 +1;  
    OUTPUT;  
RUN;
```

an error message is written to the log:

```
ERROR: Dataset name 'WHERE' is invalid due to system option DATASTMTCHK=ALLKEYWORDS
```

DATESTYLE

Specifies how date-like and time-like values are interpreted by the `ANYDTDTEw.`, `ANYDTDTMw.`, and `ANYDTTMEw.` informats.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	LOCALE
Option group:	INPUTCONTROL LANGUAGECONTROL
Portable	True
Restrictable	True
Saveable	True

The `ANYDTDTew.`, `ANYDTDTMw.` and `ANYDTTMEw.` informats convert into a numeric date value any input that has a format that looks like a date, time or datetime. The value you enter for this system option specifies how dates read into a program are interpreted for these informats. For example, `10/11/12` has different meanings depending on locale; 10 November 2012 in the UK, or 11 October 2012 in the US.

DMY

Date-like values are interpreted by the informats as day, month, year.

DYM

Date-like values are interpreted by the informats as day, year, month.

LOCALE

Date-like values are interpreted by the informats according to the current locale for the WPS session. The current locale is specified when the session starts based on the setting of

MDY

Date-like values are interpreted by the informats as month, day, year.

MYD

Date-like values are interpreted by the informats as month, year, day.

YDM

Date-like values are interpreted by the informats as year, day, month.

YMD

Date-like values are interpreted by the informats as year, month, day.

Example

In this example, the `OPTIONS` statement is used to specify that input dates are interpreted as day, month, and year.

```
OPTIONS DATESTYLE = MDY;
DATA _NULL_;
  INPUT dt ANYDTDT9.;
  PUT 'Numeric output = ' dt;
  PUT 'Reformatted output based on numeric date:' dt WORDDATX18.;
CARDS;
12/10/18
;
```


This produces the following output:

```
Numeric output = 21528
Reformatted output based on numeric date: 10 December 2018
```

The input data could be read as either the 12th of October in UK date format, or the 10th of December in US date format. However, because the system option is set as `MDY`, the input data is assumed to be in month, day, year format (US date format). The `ANYDTDTTE9 . informat` therefore returns a numeric date based on the US date format. The numeric date is then output as a formatted date string that has the equivalent US date, as shown in the line that begins `Reformatted output`, where the date is 10 December 2018.

DYNAMICNOBS

Specifies when the `NOBS` variable is set.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NODYNAMICNOBS</code>
Option group:	<code>INPUTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

A variable can be specified to the `NOBS` keyword into which is written the number of observations in a dataset. You can specify that the value for the `NOBS` variable is set:

- Only once, when the `DATA` step is run.
- When the `DATA` step is run, and then when a `SET` or `MODIFY` statement in the `DATA` step is executed.

DYNAMICNOBS

Set the `NOBS` variable when the `DATA` step is run, and when `SET` or `MODIFY` is executed.

NODYNAMICNOBS

Set `NOBS` variable only when the `DATA` step is parsed.

Basic example

In this example, the `OPTIONS` statement is used to specify that `NOBS` is only set when the `DATA` step is run.

```
OPTIONS NODYNAMICNOBS;
```

INVALIDDATA

This system option is provided for compatibility only, and has no effect in WPS.

➤ **INVALIDDATA** ➤ = ➤ *character* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	.
Maximum length:	1
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

character

For compatibility only.

S

Specifies the length of source statements and data lines.

➤ **S** ➤ = ➤ *line-length* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	2147483647
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

line-length

The length of source statements and data lines in the program. This can be specified as:

- A number; for example, if you enter 80, line consists of 80 characters.
- A number of lines multiplied by 1,024, or by 1,024², by appending K or M to the value, respectively. For example, if the value is 0.1K, the line length is 102 characters.
- MAX – the maximum supported value.

If *line-length* is set to 0 (zero), the length of source statements and data lines is unlimited.

If you set this system option and the length of one or more lines in the source exceeds the value specified, the program does not execute properly and errors are generated because, for example, a keyword might be truncated or the line termination not found.

If you truncate data lines using this option, and the informat on your INPUT statement specify a length greater than the truncated line, errors occur because the INPUT reads past the end of line.

Note:

By default, various ODS statements run before a program starts. Therefore, if you set this system option, the default ODS statements are not affected by the option. However, in Workbench, options remain set until you set a new value or restart the server. If the value of this option is set to a value smaller than the longest ODS statement, then, when the next program is run, that ODS statement fails unless the option is reset or Workbench restarted.

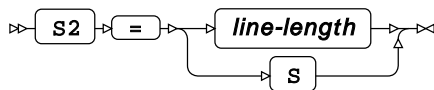
Example

In this example, the OPTIONS statement is used to specify the line length for source and data lines.

```
OPTIONS S = 200;
```

S2

Specifies the length of secondary source statements, such as those in included files.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

line-length

The length of secondary source statements.

If *line-length* is set to 0 (zero), the length of secondary source statements is unlimited.

S

Use the value assigned to the *S* system option as the maximum line-length. This is the default.

If you truncate data lines using this option, and the informat on your `INPUT` statement specify a length greater than the truncated line, errors will occur because the `INPUT` reads past the end of line.

Example

In this example, the `OPTIONS` statement is used to specify the line length for secondary source.

```
OPTIONS S2 = 500;
```

SEQ

Specifies the number of digits to remove from the numeric part of the sequence number field in lines of source.

```
SEQ = numeric-length
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	8
Minimum value:	1
Maximum value:	8
Option group:	<code>INPUTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

Use this option to remove sequence numbers from source codes.

numeric-length

The number of digits. This can be specified as:

- A number; for example, you can enter 4.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

If the *numeric-length* is greater than 0 (zero), and sequence numbers are found, the specified number of digits is removed from the sequence number.

Example

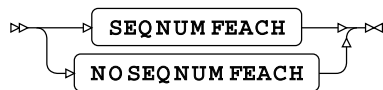
In this example, the `OPTIONS` statement is used to specify that sequence numbers over four digits long are removed.

```
OPTIONS SEQ = 4
```

Assuming the sequence numbers in subsequent programs to be four digits long, then all sequence numbers are removed.

SEQNUMFEACH

Specifies that sequence numbers are checked and removed on a per line basis.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSEQNUMFEACH</code>
Option group:	<code>INPUTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

SEQNUMFEACH

Each source input line is checked to see if it has a sequence number. If it does, then the number of digits specified by `SEQ` are removed from the sequence number.

NOSEQNUMFEACH

The first source input line is checked for a sequence number. If it has one, then all lines are assumed to have sequence numbers, and the number of digits specified by `SEQ` are removed as sequence numbers. If the first source line does not have a sequence number, all lines are assumed to not have sequence numbers.

Example

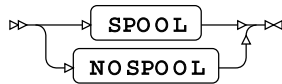
In this example, the `OPTIONS` statement is used to specify that sequence numbers are four digits long and that the presence or absence of sequence numbers is checked only at the first source line.

```
OPTIONS SEQ = 4 NOSEQNUMFEACH;
```

Assuming the sequence numbers in subsequent source lines to be four digits long, and that the first line contains a sequence number, then all sequence numbers are removed.

SPOOL

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSPOOL
Option group:	INPUTCONTROL
Portable	False
Restrictable	True
Saveable	False

SPOOL

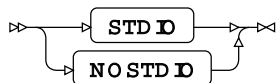
For compatibility only.

NOSPOOL

For compatibility only.

STDIO

Specifies the default streams used for input, output, and error reporting.



Valid in:	Configuration file and command line.
Default:	NOSTDIO
Option group:	INPUTCONTROL
Portable	False
Restrictable	False
Saveable	False

STDIO

Use standard streams:

- Input – standard input (stdin)
- Output – standard output (stdout)
- Error reporting – standard error (stderr)

NOSTDIO

Use the values set for the corresponding system options:

- Input – `SYSIN`
- Output – `LOG`
- Error reporting – `PRINT`

Example

In this example, the system option is specified on the command line, and sets the input, output and error reporting streams to the values of the `SYSIN`, `LOG` and `PRINT` system options.

```
WPS c:\temp\test2.wps -NOSTDIO;
```

VBUFSIZE

Specifies the size of the buffer used when executing a view in parallel.

➤ **VBUFSIZE** ➤ = ➤ *buffer-size* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	65536
Minimum value:	0
Maximum value:	Maximum integer value supported by the system.
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

buffer-size

The maximum buffer size available, in bytes. This can be specified as:

- The number of bytes; for example, you can enter `21000`.
- The number of kibibytes, mebibytes or gibibytes, by appending `K`, `M` or `G` to the value, respectively. For example, if the value is `0.5G`, the buffer size is 0.5 GiB.

- The number of bytes specified in hexadecimal, by appending `x`. For example, if the value is `BBC2X`, the buffer size is 48066 bytes.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

The value of the `OBSBUF` dataset option overrides this system option.

Example

In this example, the `OPTIONS` statement is used to specify that the view buffer is 0.25 mebibytes.

```
OPTIONS VBUF = 0.25M;
```

XCMD

Specifies whether the `x` statement is available for use.



Valid in:	Configuration file and command line.
Default:	NOXCMD
Option group:	INPUTCONTROL
Portable	False
Restrictable	True
Saveable	False

XCMD

The `x` statement is available.

NOXCMD

The `x` statement is not available.

The `x` global statement enables you to run operating system commands from a SAS language program.

Example

In this example, the system option is specified on the command line, and disables the `x` statement.

```
WPS c:\temp\testx.wps NOXCMD
```


The file `testx.wps` contains only one line, the statement `X DIR`. However, the statement cannot run because `NOXCMD` has been specified. The log contains the following messages:

```
ERROR: The X statement is disabled by the System Option "NOXCMD"
ERROR: Error printed on page 1
```

YEARCUTOFF

Specifies the starting year from which two-digit years in functions and formats are calculated.

```
>> YEARCUTOFF = year-value <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	1920
Minimum value:	1582
Maximum value:	19900
Option group:	INPUTCONTROL
Portable	True
Restrictable	True
Saveable	True

year-value

The starting year.

The default year cut off is 1926, meaning that any two-digit year is assumed to fall within the century from 1926 to 2026. The two-digit year 25 is, therefore, 2025, while the two-digit year 30 is 1930.

Example

In this example, the `OPTIONS` statement is used to specify the starting year for two-digit years.

```
OPTION YEARCUTOFF=1900;
DATA _NULL_;
  x = '16-JUN-02'D;
  PUT x DATE11.;
RUN;
```

This writes the following to the log:

```
16-JUN-1902
```

If the option had not been set, the default year is used as the cut off, and the result is:

```
16-JUN-2002
```

INSTALL group system options

System options that specify the location of installed files.

FontCACHEDIR

Specifies the location of the WPS font cache directory, used by ODS to cache fonts.

⇒ **FontCACHEDIR** ⇒ = ⇒ *pathname* ⇒

Valid in:	Configuration file and command line.
Maximum length:	32767
Option group:	INSTALL
Portable	False
Restrictable	True
Saveable	True

The specified directory is used to cache fonts required for ODS PDF fonts.

pathname

The path to the location.

This location is only used if necessary to cache the fonts. Some operating systems automatically set up cache when required, and this is used as a backup location.

Example

In this example, the system option is specified on the command line:

```
WPS c:\temp\print.wps -FontCACHEDIR c:\temp\fontcache
```

PATH

Specifies paths to the location of user-provided extensions for user programs.

⇒ **PATH** ⇒ = ⇒ *filepath* ⇒

Valid in:	Configuration file and command line.
Maximum length:	32000
Option group:	INSTALL

Portable	False
Restrictable	True
Saveable	False

filepath

The pathname of the location containing the user-provided extensions.

This system option enables you to start WPS and find executable components located elsewhere.

Example

In this example, the option is specified on the command line.

```
wps c:\temp\noo.wps -path 'c:\temp'
```

The folder `c:\temp` is searched for any user-provided extensions required by the program `nnoo.wps`.

USSWPSHOME

Specifies the location of the USS WPS installation.

```
>> USSWPSHOME = >> pathname <<
```

Valid in:	Configuration file and command line.
Maximum length:	32767
Option group:	INSTALL
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

The location of the USS WPS installation is required for some MVS WPS functionality to complete successfully, such as using outline fonts for charting, graphing, and generating reports in PDF files.

pathname

The path to the location of the USS WPS installation.

Any job that uses ODS PDF must be run with this option set to the appropriate location.

Note:

Users are prompted to provide this system option if they run the `@fontcfg.cntrl` job, which is part of the installation process on z/OS.

Example

In this example, the option is specified on the command line.

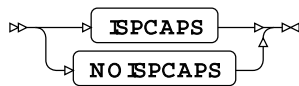
```
/u/products/worldprogramming/bin/wps -usswpshome /u/products/worldprogramming  
script.wps
```

ISPF group system options

System options that control ISPF operations and specify variables for ISPF.

ISPCAPS

Specifies whether to convert to upper case any lower case characters in arguments to the ISPF call routines.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPCAPS
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ISPCAPS

Convert characters.

NOISPCAPS

Do not convert characters.

Example

In this example, the `OPTIONS` statement is used to specify that lower-case characters are not converted to upper case characters in the ISPF call routines.

```
OPTIONS NOISPCAPS;
```

ISPCHARF

Specifies whether to convert printable characters in arguments to the ISPF call routines using their associated formats and informats.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPCHARF
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ISPCHARF

Convert printable characters.

NOISPCHARF

Do not convert printable characters.

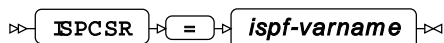
Example

In this example, the `OPTIONS` statement is used to specify that printable characters in arguments in the ISPF call routines cannot be converted using formatting.

```
OPTIONS NOISPCHARF;
```

ISPCSR

Specifies a variable that will be used by the ISPF interface to store the name of a variable whose value is found to be invalid.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False

Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ispf-varname

The name of the variable.

Example

In this example, the `OPTIONS` statement is used to specify the name of a variable used to store the name of a variable whose value is found to be invalid..

```
OPTIONS ISPCSR = INVSTR;
```

ISPEXECV

Specifies an ISPF variable that when accessed, starts an ISPF service.

```
>> ISPEXECV = ispf-varname <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ispf-varname

The name of the variable.

Example

In this example, the `OPTIONS` statement is used to specify the name of a variable.

```
OPTIONS ISPEXECV = ISTRT;
```

ISPMISS

Specifies a value that is assigned to WPS character variables that have been defined in ISPF when the associated ISPF variable has length zero.

➤ **ISPMISS** ➤ = ➤ *character* ➤

Valid in: `OPTIONS` statement, configuration file and command line.

Default:

Maximum length: 1

Option group: ISPF

Portable False

Restrictable True

Saveable True

Supported platform: z/OS for System z

character

The value to be assigned.

Example

In this example, the `OPTIONS` statement is used to specify the value to use.

```
OPTIONS ISPMISS = *;
```

ISPMMSG

Specifies the name of an ISPF variable that contains a message identifier that is accessed when a variable is found to be invalid.

➤ **ISPMMSG** ➤ = ➤ *ispf-varname* ➤

Valid in: `OPTIONS` statement, configuration file and command line.

Default: <empty-string>

Maximum length: 8

Option group: ISPF

Portable False

Restrictable True

Saveable True

Supported platform: z/OS for System z

ispf-varname

Name of the variable name that holds the message identifier.

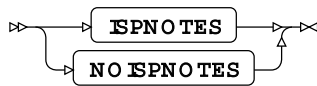
Example

In this example, the `OPTIONS` statement is used to specify the name of a variable.

```
OPTIONS ISPMMSG = INVMSGID;
```

ISPNOTES

Specifies whether to write ISPF error messages to the WPS log.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOISPNOTES`

Option group: `ISPF`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

ISPNOTES

Write ISPF error messages to the WPS log.

NOISPNOTES

Do not write ISPF error messages to the WPS log.

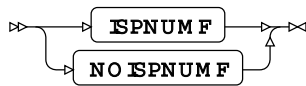
Example

In this example, the `OPTIONS` statement is used to specify that error messages are not written to the WPS log.

```
OPTIONS NOISPNOTES;
```


ISPNUMF

Specifies whether WPS converts numeric variables using associated formats and informats when the variables are used as ISPF variables.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOISPNUMF`

Option group: `ISPF`

Portable `False`

Restrictable `True`

Saveable `True`

Supported platform: `z/OS for System z`

ISPNUMF

Convert numeric values.

NOISPNUMF

Do not convert numeric values.

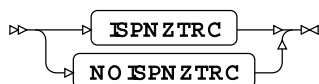
Example

In this example, the `OPTIONS` statement is used to specify that numeric variables are not converted.

```
OPTIONS ISPCSR = NOISPNUMF;
```

ISPNZTRC

Specifies whether non-zero ISPF service return codes are written to the WPS log.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOISPNZTRC`

Option group: `ISPF`

Portable `False`

Restrictable `True`

Saveable: True
Supported platform: z/OS for System z

ISPNZTRC

Write non-zero ISPF service return codes to the WPS log.

NOISPNZTRC

Do not write non-zero ISPF service return codes to WPS log.

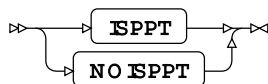
Example

In this example, the `OPTIONS` statement is used to specify that non-zero ISPF service return codes are not written to the WPS log.

```
OPTIONS NOISPNZTRC;
```

ISPPT

Specifies whether ISPF parameter pointers and lengths are written to the WPS log.



Valid in: `OPTIONS` statement, configuration file and command line.
Default: `NOISPPT`
Option group: `ISPF`
Portable: False
Restrictable: True
Saveable: True
Supported platform: z/OS for System z

ISPPT

Write ISPF parameter pointers and lengths to the WPS log.

NOISPPT

Do not write ISPF parameter pointers and lengths to the WPS log.

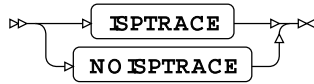
Example

In this example, the `OPTIONS` statement is used to specify that ISPF parameter pointers and lengths are not written to the WPS log.

```
OPTIONS NOISPPT;
```

ISPTRACE

Specifies whether ISPF parameter lists and service return codes are written to the WPS log.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPTRACE
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ISPTRACE

Write parameter lists and service return codes to the WPS log.

NOISPTRACE

Do not write parameter lists and service return codes to the WPS log.

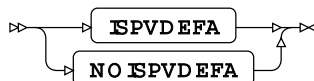
Example

In this example, the `OPTIONS` statement is used to specify that ISPF parameter lists and service return codes are not written to the WPS log.

```
OPTIONS NOISPTRACE;
```

ISPVDEFA

Specifies whether current WPS variables are identified to ISPF via the VDEFINE user exit.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPVDEFA
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True

Supported platform: z/OS for System z

ISPVDEFA

Identify variables to ISPF using VDEFINE user exit.

NOISPVDEFA

Do not identify variables to ISPF using VDEFINE user exit.

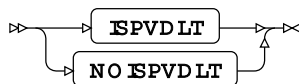
Example

In this example, the `OPTIONS` statement is used to specify that variables are not identified to ISPF using VDEFINE.

```
OPTIONS NOISPVDEFA;
```

ISPVDLT

Specifies whether the VDELETE service automatically deletes a variable from ISPF before it is defined using the VDEFINE service.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOISPVDLT`

Option group: `ISPF`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: z/OS for System z

ISPVDLT

Delete the variable automatically.

NOISPVDLT

Do not delete the variable automatically.

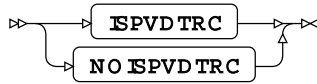
Example

In this example, the `OPTIONS` statement is used to specify that the VDELETE command is not issued automatically.

```
OPTIONS NOISPVDLT;
```

ISPVDTRC

Specifies whether to write a message to the WPS log on each call to VDEFINE.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPVDTRC
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ISPVDTRC

Write a note to the WPS log.

NOISPVDTRC

Do not write a note to the WPS log.

For each call to VDEFINE, a message is written to the WPS log.

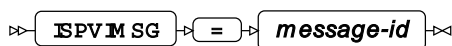
Example

In this example, the `OPTIONS` statement is used to specify that calls to VDEFINE are noted in the WPS log.

```
OPTIONS NOISPVDTRC;
```

ISPVMSG

Specifies the ISPF message identifier to be set by the VDEFINE user exit when an informat for a variable returns an error code.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF

Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

message-id

The message identifier to be set by the VDEFINE user exit.

This system option enables you to specify the message identifier that is returned by the VDEFINE user exit returns to ISPF if an informat cannot define the input (for example, if the input is invalid).

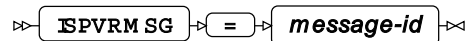
Example

In this example, the `OPTIONS` statement is used to specify the message identifier.

```
OPTIONS ISPVMSG = EXID;
```

ISPVRMSG

Specifies the ISPF message identifier to be set by the VDEFINE user exit when a variable has a null value.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

message-id

The message identifier to be set by the VDEFINE user exit.

Example

In this example, the `OPTIONS` statement is used to specify a message identifier set by the VDEFINE user exit for a null value.

```
OPTIONS ISPVRMSG = NLVID ;
```

ISPVTMSG

Specifies the ISPF message identifier to be set by the VDEFINE user exit when the ISPVTRAP option is in effect.

➤ **ISPVTMSG** ➤ = ➤ *message-id* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

message-id

The message identifier to be set by the VDEFINE user exit.

Example

In this example, the `OPTIONS` statement is used to set a message identifier.

```
OPTIONS ISPVTMSG = VTMSGID;
```

ISPVTNAM

Restricts the information returned by the `ISPVTRAP` option to a specified variable only.

➤ **ISPVTNAM** ➤ = ➤ *variable-name* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

variable-name

The variable for which to display information.

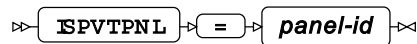
Example

In this example, the `OPTIONS` statement is used to restrict information displayed by the `ISPVTRAP` option to the variable `ISID`.

```
OPTIONS ISPVTNAM = ISID;
```

ISPVTPNL

Specifies which ISPF panel the VDEFINE user exit displays when the `ISPVTRAP` option is in effect.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

panel-id

The identifier of the panel to display.

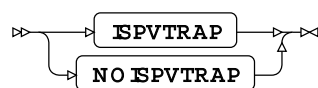
Example

In this example, the `OPTIONS` statement is used to specify the ISPF panel to display.

```
OPTIONS ISPVTPNL = PAN1ONTR;
```

ISPVTRAP

Specifies whether the VDEFINE user exit writes debugging information to the WPS log each time it is entered.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOISPVTRAP
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

ISPVTRAP

Write debugging information to the WPS log.

NOISPVTRAP

Do not write debugging information to the WPS log.

You can display information for only one variable by specifying that variable in the `ISPVTNAM` [\(page 193\)](#) system option.

Example

In this example, the `OPTIONS` statement is used to specify that the `VDEFINE` user exit does not write debugging information to the WPS log each time it is entered.

```
OPTIONS NOISPVTRAP;
```

ISPVTVARS

Specifies the prefix for the ISPF variables to be set by the `VDEFINE` user exit when the `ISPVTRAP` option is in effect.

```
>> [ISPVTVARS] [=] [prefix] <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	7
Option group:	ISPF
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

prefix

The prefix for ISPF variables.

Example

In this example, the `OPTIONS` statement is used to specify a prefix.

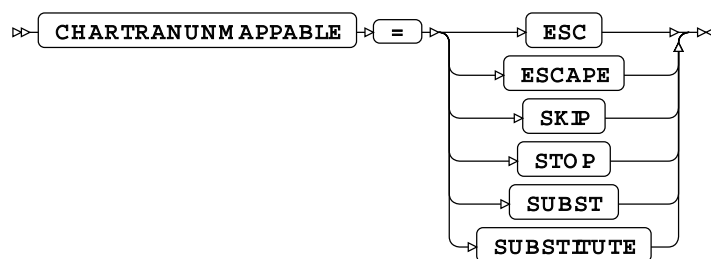
```
OPTIONS ISPVTVARS = XVIAVD;
```

LANGUAGECONTROL group system options

System options that control encoding, language and date formats.

CHARTRANUNMAPPABLE

Specifies what happens when a character cannot be mapped in the session character set.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `SUBSTITUTE`

Option group: `LANGUAGECONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

This system option enables you to specify what happens when transcoding a character from one character set to another character when that character cannot be represented in the target character set.

ESC

The character is replaced with the Unicode code point for the character.

ESCAPE

The character is replaced with the Unicode code point for the character.

SKIP

The character is skipped.

STOP

The program stops, and a message is written in the log.

SUBST

The character is replaced with the substitute character.

SUBSTITUTE

The character is replaced with the substitute character.

Example – skipping unmappable characters

In this example, the `OPTIONS` statement is used to specify that unmappable characters are skipped.

```
OPTION CHARTRANUNMAPPABLE = SKIP;
```

The following program is then run:

```
FILENAME test "c:\temp\outtext.txt";  
DATA _null_;  
  file test;  
  format result $20.;  
  result = kCVT('ca+food','utf8','latin1');  
  put result;  
RUN;
```

The resulting file contains the text `cafood`.

Example – replacing unmappable characters with substitute

In this example, the `OPTIONS` statement is used to specify that unmappable characters are replaced with the substitution character.

```
OPTION CHARTRANUNMAPPABLE = SUBST;
```

The following program is then run:

```
FILENAME test "c:\temp\outtext.txt";  
DATA _NULL_;  
  FILE test;  
  FORMAT result $20.;  
  result = KCVT('ca+food','utf8','latin1');  
  PUT result;  
RUN;
```

The resulting file contains the text `ca food`, where the represents the substitution character (`x1A` in Latin1).

Example – replacing unmappable characters with escape

In this example, the `OPTIONS` statement is used to specify that unmappable characters are replaced with the Unicode code point for escape.

```
OPTION CHARTRANUNMAPPABLE = SUBST;
```

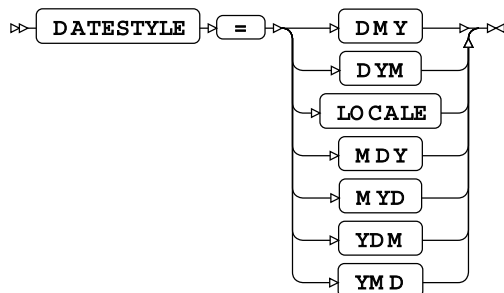
The following program is then run:

```
FILENAME test "c:\temp\outtext.txt";
DATA _null_;
  format result $14.;
  file test;
  result = KCVT('ca†food','utf8','latin1');
  put result;
RUN;
```

The resulting file contains the text `ca{U+FFFD} food`, where `{U+FFFD}` is the code point for the `†`.

DATESTYLE

Specifies how date-like and time-like values are interpreted by the `ANYDTDTEw.`, `ANYDTDTMw.`, and `ANYDTTMEw.` informats.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `LOCALE`

Option group: `INPUTCONTROL`
`LANGUAGECONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

The `ANYDTDTEw.`, `ANYDTDTMw.` and `ANYDTTMEw.` informats convert into a numeric date value any input that has a format that looks like a date, time or datetime. The value you enter for this system option specifies how dates read into a program are interpreted for these informats. For example, `10/11/12` has different meanings depending on locale; 10 November 2012 in the UK, or 11 October 2012 in the US.

DMY

Date-like values are interpreted by the informats as day, month, year.

DYM

Date-like values are interpreted by the informats as day, year, month.

LOCALE

Date-like values are interpreted by the informats according to the current locale for the WPS session. The current locale is specified when the session starts based on the setting of

MDY

Date-like values are interpreted by the informats as month, day, year.

MYD

Date-like values are interpreted by the informats as month, year, day.

YDM

Date-like values are interpreted by the informats as year, day, month.

YMD

Date-like values are interpreted by the informats as year, month, day.

Example

In this example, the `OPTIONS` statement is used to specify that input dates are interpreted as day, month, and year.

```
OPTIONS DATESTYLE = MDY;
DATA _NULL_;
  INPUT dt ANYDTDTE9.;
  PUT 'Numeric output = ' dt;
  PUT 'Reformatted output based on numeric date:' dt WORDDATX18.;
CARDS;
12/10/18
;
```

This produces the following output:

```
Numeric output = 21528
Reformatted output based on numeric date:  10 December 2018
```

The input data could be read as either the 12th of October in UK date format, or the 10th of December in US date format. However, because the system option is set as `MDY`, the input data is assumed to be in month, day, year format (US date format). The `ANYDTDTE9.` informat therefore returns a numeric date based on the US date format. The numeric date is then output as a formatted date string that has the equivalent US date, as shown in the line that begins `Reformatted output`, where the date is 10 December 2018.

DDEXLANG

Specifies the DDE triplet characters accessed through the `FILENAME DDEX` command.

» `DDEXLANG` = *supported-language* «

Valid in:	OPTIONS statement, configuration file and command line.
Default:	ENGLISH
Option group:	LANGUAGECONTROL
Portable	True
Restrictable	False
Saveable	True

supported-language

A string that specifies a corresponding, supported language, and must be one of:

- AFRIKAANS
- CATALAN
- CROATIAN
- CZECH
- DANISH
- DUTCH
- ENGLISH
- FINNISH
- FRENCH
- GERMAN
- HUNGARIAN
- ITALIAN
- LOCALE
- MACEDONIAN
- NORWEGIAN
- POLISH
- PORTUGUESE
- RUSSIAN
- SLOVENIAN
- SPANISH
- SWEDISH
- SWISS_FRENCH

- SWISS_GERMAN

A DDE triplet is the three-part specification of the DDE external file; for example, `somepath/afilename.xls/r1c1:r32c1`.

Example

In this example, the `OPTIONS` statement is used to specify the language used for DDE triplets.

```
OPTIONS DDEXLANG = FRENCH;
```

DFLANG

Specifies the language for `EURDF` date and time formats.

➤ **DFLANG** ➤ = ➤ ***format-language*** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	ENGLISH
Option group:	LANGUAGECONTROL
Portable	True
Restrictable	False
Saveable	True

format-language

A string that specifies a corresponding, supported language, and must be one of:

- AFRIKAANS
- CATALAN
- CROATIAN
- CZECH
- DANISH
- DUTCH
- ENGLISH
- FINNISH
- FRENCH
- GERMAN
- HUNGARIAN
- ITALIAN
- LOCALE

- MACEDONIAN
- NORWEGIAN
- POLISH
- PORTUGUESE
- RUSSIAN
- SLOVENIAN
- SPANISH
- SWEDISH
- SWISS_FRENCH
- SWISS_GERMAN

The EURDF date and time formats are international date formats. These are described in *International date formats*, and have a form that begins with `xxxDF`; for example, `xxxDFDDw.` or `xxxDFDNw.`.

The EURDF variant of this format does not indicate a specific language; the language for the format is instead specified by this system option.

Example

In this example, the `OPTIONS` statement is used to specify that Russian is the language that applies to EURDF formats.

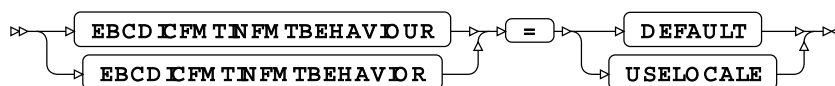
```
OPTION DFLANG = RUSSIAN;  
DATA _NULL_;  
d=MDY(12, 5, 2016);  
  PUT "Russian " d EURDFDD10.;  
RUN;
```

The results are written to the log.

```
RUSSIAN 05.12.2016
```

EBCDICFMTINFMTBEHAVIOUR

Specifies whether the `$EBCDIC` format and informat use the default behaviour, or the EBCDIC encoding for the current locale.



Valid in: Configuration file and command line.

Default: DEFAULT

Option group: LANGUAGECONTROL

Portable: False

Restrictable False

Saveable True

DEFAULT

Use the default behaviour for the `$EBCDIC` format and informat. This uses predefined character encodings (depending on the operating system).

USELOCALE

Use the EBCDIC encoding for the current locale for the `$EBCDIC` format and informat.

Example

In this example, the system specifies that any variable specified with the `$EBCDIC` informat or format is formatted using the local EBCDIC encoding, rather than the predefined encoding.

```
wps c:\temp\test.wps -EBCDICFMTINFMTBEHAVIOUR USELOCALE
```

ENCODING

Specifies the default character encoding for the session.

➤ **ENCODING** = *encoding-name* ➤

Valid in: Configuration file and command line.

Maximum length: 32

Option group: LANGUAGECONTROL

Portable False

Restrictable True

Saveable False

encoding-name

The name of the character encoding to use. The value can be the name or alternative name described in *Encoding Values*.

Example

In this example, the option is specified on the command line.

```
wps c:\temp\sadd.wpd -encoding us-ascii
```

The program `sadd.wps` is run using the US-ASCII encoding.

LOCALE

Specifies the current locale for the WPS session.

➤ **LOCALE** ➤ = ➤ **current-locale** ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32
Option group:	LANGUAGECONTROL
Portable	False
Restrictable	True
Saveable	True

current-locale

The locale to use. For a list of supported locales, see *Locale values*. You can specify either the LOCALE value, or the PXLOCALE value.

Example

In this example, the OPTIONS statement is used to specify that the locale is British English.

```
OPTIONS LOCALE = English_UnitedKingdom;
```

You can also use the PXLOCALE value:

```
OPTIONS LOCALE = en_GB;
```

NLSCOMPATMODE

Specifies that programs represent variant characters in their EBCDIC 1047 code page positions.

➤ **NLSCOMPATMODE** ➤
➤ **NONLSCOMPATMODE** ➤

Valid in:	Configuration file and command line.
Default:	NONLSCOMPATMODE
Option group:	LANGUAGECONTROL
Portable	True
Restrictable	True
Saveable	False

Older programs written in the language of SAS use characters from the EBCDIC 1047 code page. Later versions of the language of SAS enabled the use of other character encodings. Some characters, known as the variant characters, might occupy different positions in those code pages, or might not be present in EBCDIC 1047 at all. Examples of variant characters include the commercial at (@) symbol, and various parentheses (} and]).

For example, if you run a program created on a system using the EBCDIC 1047 code page on a system that uses EBCDIC 0273, and that program uses the @ character, that character is instead represented as \$. This might cause the program to fail.

By specifying this system option, variant characters are represented as if they are EBCDIC 1047 characters, whatever the character set of the system on which the program is running.

NLSCOMPATMODE

Variant characters are represented using the EBCDIC 1047 code page.

NONLSCOMPATMODE

Variant characters are represented using the session encoding code page.

Example

In this example, the system option is specified on the command line:

```
wps c:\temp\test.wps -NLSCOMPATMODE
```

The program `test.wps` in this example contains invariant characters. These are represented using characters on the EBCDIC 1047 code page.

TRANTAB

Specifies translation tables that control translation of character sets.

➤ **TRANTAB** ➤ = ➤ (➤ *table-name* ➤) ➤

Valid in: Configuration file and command line.

Maximum length: 1024

Option group: LANGUAGECONTROL

Portable: True

Restrictable: False

Saveable: True

table-name

A list of translation table names.

Translation tables names are specified in parentheses, separated by commas and positional. Each position in the list specifies a translation table that has a particular function, such as translating between the local character set and the transport character set, or between lower-case and upper-case character sets. If you do not specify a particular translation table, you must still include the comma for that position; for example:

```
TRANSTAB=( , , WLT1_UCS, WLT1_LCS, , , , , );
```

There can be no more than ten translation tables. The translation table specified at each position has the following function:

Position	The translation table is used to...
1	Translate from the local encoding to a transport encoding. This might be required for example, for WPS Communicate to connect to a remote server.
2	Translate from the transport encoding to a local encoding.
3	Translate an encoding with lower-case characters to an encoding with upper-case characters.
4	Translate an encoding with upper-case characters to an encoding with lower-case characters.
5	Provide bit fields that define character attributes for every character in the local encoding. For example, a field might define whether a character is upper case, is alphabetic, is non-printing, and so on.
6	Translate an encoding from local encoding to the compiler encoding. The compiler encoding is EBCDIC 1047 on z/OS and Latin1 on other computers.

Positions 7-10 are not currently used, and are provided for compatibility.

WPS provides various SAS language compatible translation tables in the `SASHELP` library, in `LOCALE`.

By default, the translation tables are specified as follows:

```
TRANSTAB=( , , WLT1_UCS, WLT1_LCS, WLT1_CCL, , , , , )
```

If you do not define a translation table at particular position, the default translation is used. For example, if you specify:

```
TRANSTAB=( , , myTT1_UCS, MyTT2_LCS, , , , , )
```

then the translation table `WLT1_CCL` is used at position five by default., as if you had specified:

```
TRANSTAB=( , , myTT1_UCS, MyTT2_LCS, WLT1_CCL, , , , , )
```

Example

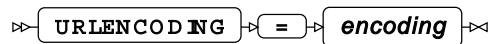
In this example, the system option is specified on the command line:

```
wps c:\temp\noo.wps -TRANTAB (,,E142_LCS,E142_LCS,,,,)
```

The translation tables used for case translations are the EBCDIC 1142 Denmark/Norway tables found in the `SASHELP LOCALE` library member.

URLENCODING

Specifies how the strings specified to the `URLENCODE` and `URLDECODE DATA` step functions are encoded and decoded.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	SESSION
Option group:	LANGUAGECONTROL
Portable	True
Restrictable	True
Saveable	True

encoding

The encoding:

- SESSION
- UTF8

Example

In this example, the `OPTIONS` statement is used to specify that values specified to the `URLENCODE` and `URLDECODE` functions are encoded and decoded as UTF-8 characters:

```
OPTIONS URLENCODING = UTF8;
```

LOG_LISTCONTROL group system options

System options that control the layout of both logs and listings.

DATE

Specifies whether the date and time are printed at the top of each page a log or listing.



Valid in: OPTIONS statement, configuration file and command line.

Default: NODATE

Option group: LOGCONTROL
LISTCONTROL
LOG_LISTCONTROL

Portable True

Restrictable True

Saveable True

DATE

Print the date and time at the top.

NODATE

Do not print the date and time at the top.

Example

In this example, the `OPTIONS` statement is used to specify that no date is added to output listings.

```
OPTIONS NODATE;
```

If the following program is run:

```
LIBNAME books 'C:\temp\books';
PROC SORT DATA=books.lib_books;
BY author;

PROC REPORT DATA=_LAST_;
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";
COLUMN title;
BY author;
```

The first page of the listing has no date, only a page number.

```

                                The WPS System                                1
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor
```

If instead the option is set to `DATE` (the default), and the same program run, the first line of the listing is:

```

                                The WPS System                                1
                                15:33 Wednesday, February 27, 2019
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor

```

DETAILS

Specifies whether to provide additional details about members in data libraries through the `CONTENTS` and `DATASETS` procedures.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODETAILS`

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

If `DETAILS` is set, then information is provided in logs and listings about the numbers of observations, variables and labels in each dataset returned by the `CONTENTS` or `DATASETS` procedure. Information on members of a library is only provided when using the `CONTENTS` procedure if the `_ALL_` system variable is provided as the input dataset; for example:

```
PROC CONTENTS DATA=mydir._ALL_
```

DETAILS

Show additional details.

NODETAILS

Do not show additional details.

Example

In this example, the `OPTIONS` statement is used to specify that additional detail is written to the default outputs.

```
LIBNAME books "c:\temp\books";
OPTIONS DETAILS;
PROC CONTENTS DATA=books._ALL_;
RUN;
```

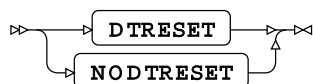
This produces the following in the `Members` section of the HTML output:

Members							
Number	Member Name	Member Type	Obs or Entries	Vars	Label	File Size	Date Last Modified
1	BOOKS	DATA	2010	7		491520	09OCT2018:15:30:00
2	BOOKS_AUTHOR_SORTED	DATA	2010	7		491520	26SEP2016:11:38:30
3	BOOKS_DATE_SORTED	DATA	2010	7		491520	06OCT2016:10:11:21
4	BOOKS_NE	DATA	1	0		8192	05OCT2018:15:11:34
5	BOOKS_OUT	DATA	2010	7		491520	05OCT2018:15:07:56
6	LIB_BOOKS	DATA	1826	10		1003520	17OCT2018:14:22:54
7	SORTEDBKS	DATA	1826	10		1003520	22OCT2018:10:46:32

The members section of the output lists all members of the specified library `books`, including datasets and catalogs. The entries `Obs or Entries`, `Vars` and `Label` have been added to the table.

DTRESET

Specifies whether to update the date and time in the titles of logs and listings when a new page is written.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODTRESET`

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

DTRESET

Update the date and time.

NODTRESET

Do not update the date and time.

If **NODTRESET**, the date and time is set to the date and time of the first page of the listing file.

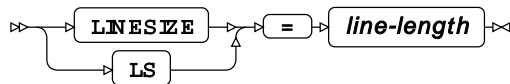
Example

In this example, the **OPTIONS** statement is used to specify that the date and time are updated each time a page of output is written.

```
OPTIONS DTRESET;
```

LINESIZE

Specifies the line length for logs and listings.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	99
Minimum value:	64
Maximum value:	256
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	False
Saveable	True

line-length

The number of characters in each line. This can be specified as:

- A number; for example, if you enter 80, the maximum line length is 80 characters.
- **MIN** – the minimum supported value.
- **MAX** – the maximum supported value.

Note:

The default line length for non z/OS operating systems is 99 characters; the default line length for z/OS operating systems is 132 characters.

Example

In this example, the `OPTIONS` statement is used to specify the line length.

```
OPTION LINESIZE = 64;
DATA _NULL_;

  PUT "=====><===== ";
RUN;
```

This produces the following output:

```
=====>
<=====
```

The output in the log has split after the 64th character, as the maximum line length is 64 characters.

MISSING

Specifies the character used to represent missing numeric values.

```
>> MISSING = character <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	.
Maximum length:	1
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	True
Saveable	True

The character specified is used to represent missing numeric values in logs, listing and datasets.

character

The character to use.

Example

In this example, the `OPTIONS` statement is used to specify that the + (plus) is used as the missing value character.

```
OPTIONS MISSING = '+';
DATA _NULL_;
  PUT x=;
RUN;
```

This produces the following output:

```
NOTE: Variable "x" may not be initialized
```

```
x=+
```

NUMBER

Specifies whether to print the page number at the top of each log and listing page.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NONNUMBER`

Option group: `LOGCONTROL`
 `LISTCONTROL`
 `LOG_LISTCONTROL`

Portable `True`

Restrictable `True`

Saveable `True`

NUMBER

Print the page number at the top of each page.

NONNUMBER

Do not print the page number at the top of each page.

Example

In this example, the `OPTIONS` statement is used to specify that page numbers are not printed at the top of each page.

```
OPTIONS NONNUMBER;  
LIBNAME books 'C:\temp\books';  
PROC REPORT DATA=books.lib_books;  
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";  
column title;  
BY author;
```

This writes the following to the first page of the log. There is no page number in the header at the top of the page.

```

The WPS System
15:33 Wednesday, February 27, 2019

----- Author=Abraham, J H -----
Title
Origins and Growth of Sociology, The

```

If the option had been set to `NUMBER`, the page header would contain a page number:

```

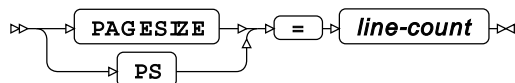
The WPS System
15:33 Wednesday, February 27, 2019
1

----- Author=Abraham, J H -----
Title
Origins and Growth of Sociology, The
1

```

PAGESIZE

Specifies the number of lines that make up a page of logs and listings.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: 55

Minimum value: 15

Maximum value: 32767

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: True

Restrictable: False

Saveable: True

line-count

The number of lines per page. This can be specified as:

- The number of lines; for example, if you enter 25, a page consists of 25 lines.
- The number of lines as a multiple of 1,024, by appending `K` to the value; for example, if the value is 0.2K, a page consists of 205 lines. Decimal values are rounded.
- The number of lines specified in hexadecimal, by appending `x`. For example, if the value is 1EX, a page consists of 30 lines.
- `MIN` – the minimum supported value.

- `MAX` – the maximum supported value.

Example

In this example, the `OPTIONS` statement is used to specify a page length of 30 lines.

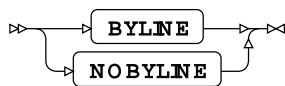
```
OPTIONS PAGESIZE = 30;
```

LISTCONTROL group system options

System options that control the appearance of output listings.

BYLINE

Specifies whether a title line is generated for each `BY` group in a listing.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOBYLINE</code>
Option group:	<code>LISTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

BYLINE

Generate a title.

NOBYLINE

Do not generate a title.

`BY` group titles are generated for listings by default. To switch them off, specify `NOBYLINE`. Using `BY` group titles can make the output clearer.

Example

In this example, the `OPTIONS` statement is used to specify that no titles are added for `BY` groups.

```
LIBNAME books 'c:\temp\books';
OPTIONS NOBYLINE;
PROC PRINT DATA = books.books;
BY author;
```

No title is written at the top of each page. A page of the output might look like this, for example:

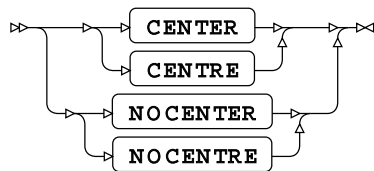
Obs	Title	Type
1	Origins and Growth of Sociology, The	Soc

When `BYLINE` is specified, the subject of the `BY` is written in the title.

----- Author=Abraham, J H -----		
Obs	Title	Type
1	Origins and Growth of Sociology, The	Soc

CENTER

Specifies whether listing output is centre-aligned on listings.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOCENTER`

Option group: `LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

CENTER

Centre listing output.

NOCENTER

Do not centre listing output. The output is left-aligned.

Example

In this example, the `OPTIONS` statement is used to specify that output is left-aligned..

```
LIBNAME books 'C:\temp\books';
OPTIONS NOCENTER;
PROC REPORT DATA=books.books;
COLUMN title;
BY author;
```

The first page of the listing output looks like this:

```
The WPS System                                16:41 Wednesday, February
 27, 2019      1

Author=Abraham, J H

Title

Origins and Growth of Sociology, The
```

If the system option is set to:

```
OPTIONS CENTER;
```

The first page of the listing output looks like this:

```

                                The WPS System  16:41 Wednesday, February
27, 2019      1

----- Author=Abraham, J H
-----

Title

Origins and Growth of Sociology, The
```

DATE

Specifies whether the date and time are printed at the top of each page a log or listing.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NODATE
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True

Restrictable True

Saveable True

DATE

Print the date and time at the top.

NODATE

Do not print the date and time at the top.

Example

In this example, the `OPTIONS` statement is used to specify that no date is added to output listings.

```
OPTIONS NODATE;
```

If the following program is run:

```
LIBNAME books 'C:\temp\books';
PROC SORT DATA=books.lib_books;
BY author;

PROC REPORT DATA=_LAST_;
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";
COLUMN title;
BY author;
```

The first page of the listing has no date, only a page number.

```

                                The WPS System                                1
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor
```

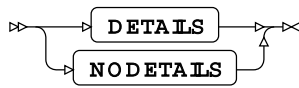
If instead the option is set to `DATE` (the default), and the same program run, the first line of the listing is:

```

                                The WPS System                                1
                                15:33 Wednesday, February 27, 2019
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor
```


DETAILS

Specifies whether to provide additional details about members in data libraries through the CONTENTS and DATASETS procedures.



Valid in: OPTIONS statement, configuration file and command line.

Default: NODETAILS

Option group: LOGCONTROL
LISTCONTROL
LOG_LISTCONTROL

Portable True

Restrictable True

Saveable True

If **DETAILS** is set, then information is provided in logs and listings about the numbers of observations, variables and labels in each dataset returned by the CONTENTS or DATASETS procedure. Information on members of a library is only provided when using the CONTENTS procedure if the `_ALL_` system variable is provided as the input dataset; for example:

```
PROC CONTENTS DATA=mydir._ALL_
```

DETAILS

Show additional details.

NODETAILS

Do not show additional details.

Example

In this example, the **OPTIONS** statement is used to specify that additional detail is written to the default outputs.

```
LIBNAME books "c:\temp\books";  
OPTIONS DETAILS;  
PROC CONTENTS DATA=books._ALL_;  
RUN;
```

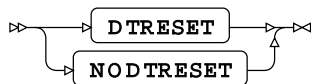
This produces the following in the `Members` section of the HTML output:

Members							
Number	Member Name	Member Type	Obs or Entries	Vars	Label	File Size	Date Last Modified
1	BOOKS	DATA	2010	7		491520	09OCT2018:15:30:00
2	BOOKS_AUTHOR_SORTED	DATA	2010	7		491520	26SEP2016:11:38:30
3	BOOKS_DATE_SORTED	DATA	2010	7		491520	06OCT2016:10:11:21
4	BOOKS_NE	DATA	1	0		8192	05OCT2018:15:11:34
5	BOOKS_OUT	DATA	2010	7		491520	05OCT2018:15:07:56
6	LIB_BOOKS	DATA	1826	10		1003520	17OCT2018:14:22:54
7	SORTEDBKS	DATA	1826	10		1003520	22OCT2018:10:46:32

The members section of the output lists all members of the specified library `books`, including datasets and catalogs. The entries `Obs or Entries`, `Vars` and `Label` have been added to the table.

DTRESET

Specifies whether to update the date and time in the titles of logs and listings when a new page is written.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODTRESET`

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

DTRESET

Update the date and time.

NODTRESET

Do not update the date and time.

If `NODTRESET`, the date and time is set to the date and time of the first page of the listing file.

Example

In this example, the `OPTIONS` statement is used to specify that the date and time are updated each time a page of output is written.

```
OPTIONS DTRESET;
```

FILESYSOUT

Specifies the default `SYSOUT` class for a printer file.

➤ **FILESYSOUT** ➤ = ➤ **output-class** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	1
Option group:	<code>LISTCONTROL</code>
Supported platform:	z/OS for System z

output-class

The default `SYSOUT` class to use.

Example

In this example, the `OPTIONS` statement is used to specify the `SYSOUT` class.

```
OPTIONS FILESYSOUT = B;
```

FORMCHAR

Specifies the output formatting characters.

➤ **FORMCHAR** ➤ = ➤ **character-list** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	--- + ---+= -\<>*
Maximum length:	20
Option group:	<code>LISTCONTROL</code>
Portable	True
Restrictable	True

Saveable True

character-list

A set of characters representing drawing primitives that in combination can delineate the lines and boxes of output tables on systems with varying levels of font support.

The position of a character in *character-list* determines what drawing element the character enables. For example, position 1 in the list is the vertical rule; position 3 is the top left corner of a box. Specifying values to this system option sets new values for all drawing primitives. Therefore, if you only specify characters in the first three positions, all subsequent positions are treated as space characters. For example, if you specify `OPTIONS FORMCHAR = ' | -+ '`, only the vertical rule, horizontal rule and top left corner drawing primitives are defined; all other drawing primitives are set to space characters. If you subsequently run the following program:

```
LIBNAME books 'C:\temp\books';
PROC REPORT DATA=books.books_out box;
COLUMN author title;
WHERE type EQ "SF";
DEFINE author / width = 30;
DEFINE title / width = 50;
```

The following listing is written; in this example, just the first few lines are shown:

```
+-----+-----+
|Author                                | Title                                |
+-----+-----+
|Adams, Douglas                        | Hitch-hiker's Guide to the Galaxy, The |
+-----+-----+
|Adams, Douglas                        | Restaurant at the End of the Universe, The |
+-----+-----+
|Adams, Douglas                        | Restaurant at the End of the Universe, The |
+-----+-----+
```

A character is written for the top left-hand corner, for vertical lines, and for horizontal lines. However, no character is written for the top right-hand corner, for the corners between rows and verticals, or for the junction of internal verticals.

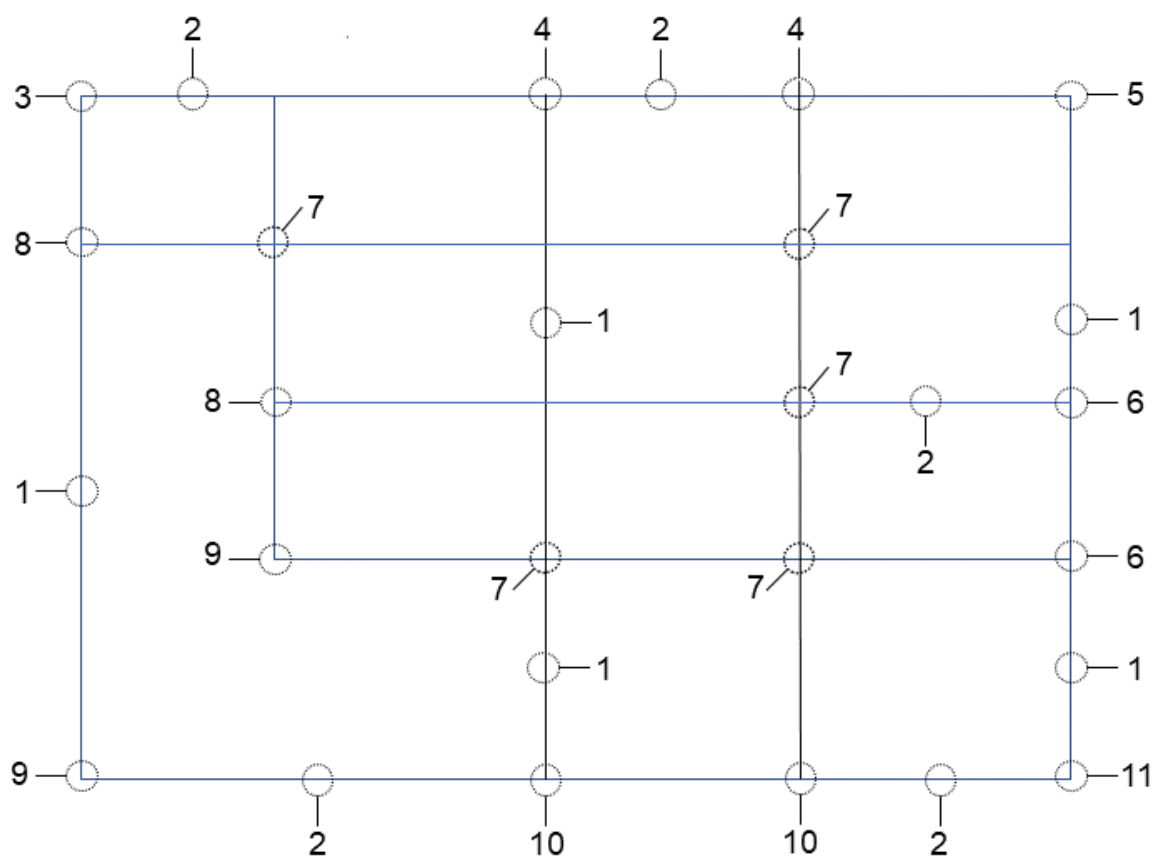
The first eleven positions of *character-list* define elements for tabulated data, and specify the characters used for corners, vertical and horizontal elements, junctions, and so on. Subsequent positions define drawing primitives used by various procedures, and the use of the characters at those positions depends on and is defined by the procedure.

The first eleven positions of *character-list* define the following elements:

Position	Element
1	Vertical rule
2	Horizontal rule
3	Top left corner
4	Top horizontal joint
5	Top right corner

Position	Element
6	Left vertical joint
7	Internal joint
8	Right vertical joint
9	Bottom left corner
10	Bottom horizontal joint
11	Bottom right corner

In the following diagram, a table has been annotated to indicate the position of elements. The numbers correspond to the elements listed above.



Example

In this example, the `OPTIONS` statement is used to specify the characters used to draw table frames.

```
OPTION PAGESIZE=20;
OPTION FORMCHAR= '|-+^+:::~+';
LIBNAME books 'C:\temp\books';
PROC REPORT DATA=books.books_out BOX;
COLUMN author title;
WHERE TYPE EQ "SF";
DEFINE author / WIDTH = 30;
DEFINE title / WIDTH = 50;
```

The `PAGESIZE` system option has also been set in this example, so that a page of the listing can be more easily shown here. The program creates a listing in which the frames of the tables are created using the specified characters:

```
+-----+-----+
| Author                | Title                               |
+-----+-----+
| Blish, James          | Day After Judgement, The           |
+-----+-----+
| Blish, James          | Black Easter                       |
+-----+-----+
| Blish, James and Knight, Damon | Torrent of Faces, A               |
+-----+-----+
| Bova, Ben             | Colony                             |
+-----+-----+
| Bova, Ben             | Mars                              |
+-----+-----+
| Bradbury, Ray         | Fahrenheit 451                     |
+-----+-----+
```

FORMDLIM

Specifies a character that delimits page breaks in listing output.

```
>> FORMDLIM = character <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code><empty-string></code>
Maximum length:	1
Option group:	<code>LISTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

character

Character to use as the page break delimiter.

The character you specify is repeated across the full width of the top of the page.

Example

In this example, the `OPTIONS` statement is used to specify that a page break delimiter is used.

```
OPTIONS FORMDELIM = "@";
```

The following is written at the top of each page:

[illegible]

LABEL

Specifies whether dataset variable labels are used in place of variable names in listings.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: NOLABEL

Option group: LISTCONTROL

Portable True

Restrictable	True
--------------	------

Saveable	True
----------	------

LABEL

Use variable labels.

NOLABEL

Do not use variable labels.

Example

In this example, the `OPTIONS` statement is used to specify that variable labels are used in listings.

```
OPTIONS label;  
PROC REPORT DATA=books.lib_books;  
COLUMN title datepurchased;  
DEFINE title/width=50;  
DEFINE datepurchased/width=20;
```

This produces a listing in which the first few lines look like this:

```

1                               The WPS System  17:45 Wednesday, February 27, 2019

                                Title                                Date book purchased

                                Hidden Histories of Science          09APR99

                                Energy and the Earth Machine        01JUN89

                                Origins and Growth of Sociology, The 01JUN86

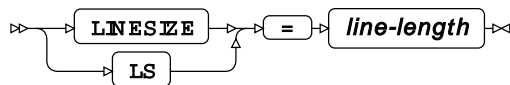
                                Chatterton                          11JAN15

```

The `datepurchased` variable name has been replaced in the listing with the variable label `Date book purchased`.

LINESIZE

Specifies the line length for logs and listings.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: 99

Minimum value: 64

Maximum value: 256

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: True

Restrictable: False

Saveable: True

line-length

The number of characters in each line. This can be specified as:

- A number; for example, if you enter 80, the maximum line length is 80 characters.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

Note:

The default line length for non z/OS operating systems is 99 characters; the default line length for z/OS operating systems is 132 characters.

Example

In this example, the `OPTIONS` statement is used to specify the line length.

```
OPTION LINESIZE = 64;
DATA _NULL_;

  PUT "=====><===== ";
RUN;
```

This produces the following output:

```
=====>
<=====
```

The output in the log has split after the 64th character, as the maximum line length is 64 characters.

LISTINGFILERECFM

Specifies the default record format (RECFM) to use for listing files.

⇒ **LISTINGFILERECFM** = *recfm* ⇐

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	VBA
Maximum length:	8
Option group:	LISTCONTROL
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

recfm

The default RECFM.

See your IBM documentation for information on RECFM formats.

Example

In this example, the `OPTIONS` statement is used to specify that the record format for a listing file is `FB`.

```
OPTIONS LISTINGFILERECFM = FB;
```

MISSING

Specifies the character used to represent missing numeric values.

➤ **MISSING** ➤ = ➤ *character* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	.
Maximum length:	1
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	True
Saveable	True

The character specified is used to represent missing numeric values in logs, listing and datasets.

character

The character to use.

Example

In this example, the `OPTIONS` statement is used to specify that the `+` (plus) is used as the missing value character.

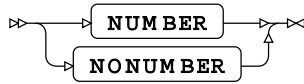
```
OPTIONS MISSING = '+';  
DATA _NULL_;  
  PUT x=;  
RUN;
```

This produces the following output:

```
NOTE: Variable "x" may not be initialized  
  
x=+
```

NUMBER

Specifies whether to print the page number at the top of each log and listing page.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NONUMBER
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	True
Saveable	True

NUMBER

Print the page number at the top of each page.

NONUMBER

Do not print the page number at the top of each page.

Example

In this example, the `OPTIONS` statement is used to specify that page numbers are not printed at the top of each page.

```
OPTIONS NONUMBER;
LIBNAME books 'C:\temp\books';
PROC REPORT DATA=books.lib_books;
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";
column title;
BY author;
```

This writes the following to the first page of the log. There is no page number in the header at the top of the page.

```

                                The WPS System
                                15:33 Wednesday, February 27, 2019

----- Author=Abraham, J H -----
Title
Origins and Growth of Sociology, The
```

If the option had been set to `NUMBER`, the page header would contain a page number:

The WPS System	1
15:33 Wednesday, February 27, 2019	
----- Author=Abraham, J H -----	
Title	
Origins and Growth of Sociology, The	1

PAGENO

Specifies the page number to use for the next page of printed output.

» `PAGENO` = `page-number` «

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	1
Maximum value:	2147483647
Option group:	<code>LISTCONTROL</code>
Portable	True
Restrictable	True
Saveable	True

page-number

The number of the next page. This can be specified as:

- The number of the page; for example, 10.
- A number of lines multiplied by 1,024, or by 1,024² by appending `K` or `M` to the value, respectively. For example, if the value is 0.5K, the next page number is 512.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

If you do set *page-number* to `MAX`, page numbering behaves in ways you might not expect. The first page number is 2147483647, the second number is -2147483648, and subsequent page numbers increment from there (-2147483647, -2147483646, and so on).

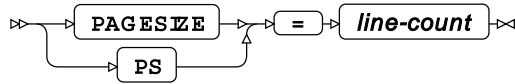
Example

In this example, the `OPTIONS` statement is used to specify the next page number in the listing.

```
OPTIONS PAGENO = 50;
```

PAGESIZE

Specifies the number of lines that make up a page of logs and listings.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	55
Minimum value:	15
Maximum value:	32767
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	False
Saveable	True

line-count

The number of lines per page. This can be specified as:

- The number of lines; for example, if you enter 25, a page consists of 25 lines.
- The number of lines as a multiple of 1,024, by appending κ to the value; for example, if the value is 0.2 κ , a page consists of 205 lines. Decimal values are rounded.
- The number of lines specified in hexadecimal, by appending \times . For example, if the value is 1EX, a page consists of 30 lines.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

Example

In this example, the `OPTIONS` statement is used to specify a page length of 30 lines.

```
OPTIONS PAGESIZE = 30;
```

LOGCONTROL group system options

System options that control the layout of output listings.

DATE

Specifies whether the date and time are printed at the top of each page a log or listing.



Valid in: OPTIONS statement, configuration file and command line.

Default: NODATE

Option group: LOGCONTROL
LISTCONTROL
LOG_LISTCONTROL

Portable True

Restrictable True

Saveable True

DATE

Print the date and time at the top.

NODATE

Do not print the date and time at the top.

Example

In this example, the `OPTIONS` statement is used to specify that no date is added to output listings.

```
OPTIONS NODATE;
```

If the following program is run:

```
LIBNAME books 'C:\temp\books';
PROC SORT DATA=books.lib_books;
BY author;

PROC REPORT DATA=_LAST_;
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";
COLUMN title;
BY author;
```

The first page of the listing has no date, only a page number.

```

                                     The WPS System                                     1
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor
  
```

If instead the option is set to `DATE` (the default), and the same program run, the first line of the listing is:

```

                                The WPS System                                1
                                15:33 Wednesday, February 27, 2019
----- Author=Ackroyd, Peter -----
Title
Chatterton
First Light
Hawksmoor
```

DETAILS

Specifies whether to provide additional details about members in data libraries through the `CONTENTS` and `DATASETS` procedures.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODETAILS`

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

If `DETAILS` is set, then information is provided in logs and listings about the numbers of observations, variables and labels in each dataset returned by the `CONTENTS` or `DATASETS` procedure. Information on members of a library is only provided when using the `CONTENTS` procedure if the `_ALL_` system variable is provided as the input dataset; for example:

```
PROC CONTENTS DATA=mydir._ALL_
```

DETAILS

Show additional details.

NODETAILS

Do not show additional details.

Example

In this example, the `OPTIONS` statement is used to specify that additional detail is written to the default outputs.

```
LIBNAME books "c:\temp\books";
OPTIONS DETAILS;
PROC CONTENTS DATA=books._ALL_;
RUN;
```

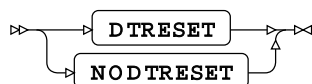
This produces the following in the `Members` section of the HTML output:

Members							
Number	Member Name	Member Type	Obs or Entries	Vars	Label	File Size	Date Last Modified
1	BOOKS	DATA	2010	7		491520	09OCT2018:15:30:00
2	BOOKS_AUTHOR_SORTED	DATA	2010	7		491520	26SEP2016:11:38:30
3	BOOKS_DATE_SORTED	DATA	2010	7		491520	06OCT2016:10:11:21
4	BOOKS_NE	DATA	1	0		8192	05OCT2018:15:11:34
5	BOOKS_OUT	DATA	2010	7		491520	05OCT2018:15:07:56
6	LIB_BOOKS	DATA	1826	10		1003520	17OCT2018:14:22:54
7	SORTEDBKS	DATA	1826	10		1003520	22OCT2018:10:46:32

The members section of the output lists all members of the specified library `books`, including datasets and catalogs. The entries `Obs or Entries`, `Vars` and `Label` have been added to the table.

DTRESET

Specifies whether to update the date and time in the titles of logs and listings when a new page is written.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODTRESET`

Option group: `LOGCONTROL`
`LISTCONTROL`
`LOG_LISTCONTROL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

DTRESET

Update the date and time.

NODTRESET

Do not update the date and time.

If **NODTRESET**, the date and time is set to the date and time of the first page of the listing file.

Example

In this example, the **OPTIONS** statement is used to specify that the date and time are updated each time a page of output is written.

```
OPTIONS DTRESET;
```

ECHOAUTO

Specifies whether to include in the log the contents of the program specified using **AUTOEXEC**.



Valid in: Configuration file and command line.

Default: NOECHOAUTO

Option group: LOGCONTROL

Portable: True

Restrictable: True

Saveable: False

The **AUTOEXEC** [↗](#) (page 95) system option enables you to specify a SAS language program that is run when a WPS session starts.

ECHOAUTO

Include the content of the **AUTOEXEC** program in the log.

NOECHOAUTO

Do not include the content of the **AUTOEXEC** program in the log.

Example

In this example, the option is specified on the command line. The program specified by **AUTOEXEC** is included in the log.

```
wps c:\temp\test2.wps -autoexec "c:\temp\noo.wps" -echoauto
```

The log contains the following:

```
NOTE: AUTOEXEC processing beginning; file is c:\temp\noo.wps
NOTE: AUTOEXEC source line
```

```

1      +
NOTE: AUTOEXEC source line
2      + DATA _NULL_;
NOTE: AUTOEXEC source line
3      + num = 1 + 2 * 3 - 6;
NOTE: AUTOEXEC source line
4      + PUT num;
NOTE: AUTOEXEC source line
5      + RUN;

1
NOTE: The data step took :
      real time : 0.013
      cpu time  : 0.000

NOTE: AUTOEXEC processing completed

1
2
3      data _null_;
4
5      put "This program runs first";
6
7      run;

This program runs first
NOTE: The data step took :
      real time : 0.011
      cpu time  : 0.000

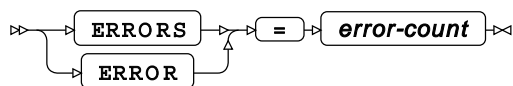
8

NOTE: Submitted statements took :
      real time : 0.108
      cpu time  : 0.046

```

ERRORS

Specifies the maximum number of observations for which error messages are output.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	20
Minimum value:	0
Maximum value:	2147483647
Option group:	ERRORHANDLING LOGCONTROL

Portable	True
Restrictable	True
Saveable	True

error-count

The number of observations for which error messages are output.

Example

In this example, the `OPTIONS` statement is used to specify that the program stops producing messages if the number of errors exceeds five.

```
OPTIONS ERRORS=5;
LIBNAME books XLSX 'c:\temp\books\books.xlsx';
DATA out;
    SET books.books1;
    IF title EQ 6 THEN OUTPUT;
RUN;
```

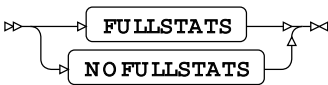
The `IF` statement attempts to compare a character value to a numeric value, which causes an error. This error is reported in the log for each observation, until the fifth is reached:

```
WARNING: Limit set by ERRORS= reached : No further messages of this type will be
printed
_N_=5 _ERROR_=1 _IORC_=0 Title=English Common Reader, The Type=History
Author=Altick, Richard D
Read=a Date_Read= Owned=n Y_Total=.
```

No more errors are reported after this observation.

FULLSTATS

Specifies whether to write more detailed performance statistics for a step on z/OS.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOFULLSTATS</code>
Option group:	<code>LOGCONTROL</code> <code>PERFORMANCE</code>
Supported platform:	z/OS for System z

FULLSTATS

Write more detailed performance statistics.

NOFULLSTATS

Do not write more detailed performance statistics.

By default, the information returned is file information and the amount of real time and processor time used to execute the step. If you specify **FULLSTATS**, additional information is provided about the EXCP count. This information is added after the data step timing; for example:

```
real time : 0.002
cpu time  : 0.001
EXCP count: 0
```

Example

In this example, the **OPTIONS** statement is used to specify full statistics are written to the log.

```
OPTIONS FULLSTATS;
```

If a program is then run, the following is written to the log:

```
The file INLINE is:

Dsnname = CJH.FULLSTAT.JOB08311.D0000101.?,
Unit =,

Volume =, Disp = NEW, Blksize=80, Lrecl=80, Recfm=FB

Creation=2018/08/01

LINE
1

LINE
2

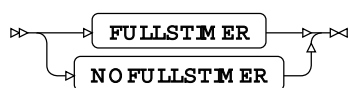
NOTE: 2 records were read from file
INLINE
NOTE: The data step took:

    real time : 0.042
    cpu time  : 0.003
    EXCP count: 0
```

The EXCP count has been added to the log.

FULLSTIMER

Specifies whether to write more detailed performance statistics for a step.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOFULLSTIMER
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

FULLSTIMER

Write additional performance statistics to the log.

NOFULLSTIMER

Do not write additional performance statistics to the log.

By default, the only information returned is the amount of real time and processor time used to execute the step; for example:

```
real time : 4.816
cpu time  : 0.015
```

If you specify **FULLSTIMER** additional information is returned:

user CPU time	The amount of taken by the processor to execute user code.
system CPU time	The amount of taken by the processor to execute operating system instructions.
Peak working set	The peak working set allocated for the process.
Current working set	The working set used by the process.
Page fault count	The number of page faults that occurred.

For example:

```
real time      : 2.419
user cpu time   : 0.000
system cpu time : 0.015
Peak working set : 25580k
Current working set : 25548k
Page fault count : 20
```

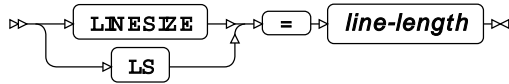
Example

In this example, the **OPTIONS** statement is used to specify that all execution information is returned.

```
OPTIONS FULLSTIMER;
```

LINESIZE

Specifies the line length for logs and listings.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	99
Minimum value:	64
Maximum value:	256
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	False
Saveable	True

line-length

The number of characters in each line. This can be specified as:

- A number; for example, if you enter 80, the maximum line length is 80 characters.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

Note:

The default line length for non z/OS operating systems is 99 characters; the default line length for z/OS operating systems is 132 characters.

Example

In this example, the `OPTIONS` statement is used to specify the line length.

```
OPTION LINESIZE = 64;  
DATA _NULL_;  
  
  PUT "=====><===== " ;  
  
RUN;
```

This produces the following output:

```
=====>  
<=====
```

The output in the log has split after the 64th character, as the maximum line length is 64 characters.

LOGPARM

Specifies log file configuration parameters.



Valid in: Configuration file and command line.

Option group: LOGCONTROL

Portable: True

Restrictable: True

Saveable: False

Valid in: Configuration file and command line

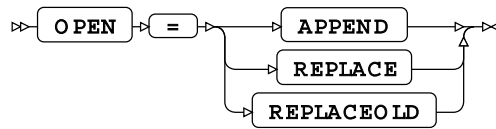
Max length: 256

parameter-options

A string specifying a list of `<option>=<value>` configuration parameters. The string can contain the following options:

OPEN

Specifies the action when a log file is opened, in the format:



APPEND

Log information is appended to the existing log file. If no log file exists, a new log file is created.

REPLACE

A new log file is created. If a log file of the same name exists, it is overwritten by the new log file.

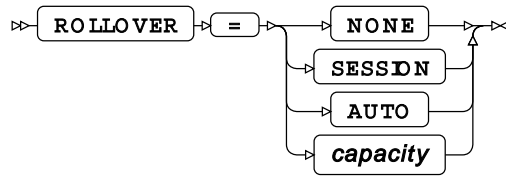
REPLACEOLD

A new log file is created when the last modification time for the existing log file is more than 24 hours earlier. If a log file of the same name exists, it is overwritten by the new log file.

If the last modification time is less than 24 hours earlier, log information is appended to the current file.

ROLLOVER

Specifies whether a new log file is created, in the format:

**AUTO**

Opens a new log file when the file name specified in the `LOG` system option changes. A file name change only occurs if the log file name contains naming directives, for example a file name of `#H-#M.log` generates a new log every minute.

NONE

Never roll over to a new log file. This is the default. If the file name specified in the `LOG` system option contains naming directives, these directives are treated as literal characters.

SESSION

Opens a new log file when the WPS process starts, and uses the same log file for the duration that the WPS process is running. If the file name specified in the `LOG` system option contains naming directives, those directives are processed.

capacity

Defines the amount of disk space for allocated for a log file. *capacity* is specified as a numeric value optionally followed by `K` (kibibyte), `M` (mebibyte), or `G` (gibibyte).

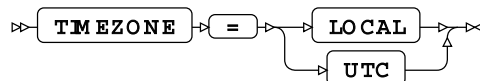
The new log file begins at the start of the next log page when the defined *capacity* is reached, but page numbers are not reset. Any *capacity* value defined for z/OS systems is an approximation.

When a log file rolls over to a new file, the existing log file name has *old* appended to the base name, for example `wpslog.txt` becomes `wpslogold.txt`

Only one *old* version of the log file is kept; if a roll-over is required and there is an existing *old* version of the log file, that file is overwritten.

TIMEZONE

Specifies the timezone used for dates and time naming directives defined in the `LOG` or `ALTLOG` file names formats.

**LOCAL**

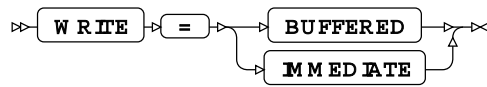
Use the local time zone for naming directives. This is the default.

UTC

Use UTC as the time zone for naming directives. Using UTC prevents log information potentially being lost at daylight saving time changes.

WRITE

Specifies the method used to write information to the log file, in the format:

**BUFFERED**

Stores information to buffer memory and writes the information to the log file when the buffer is full.

IMMEDIATE

Log information is written to the file as it is generated in WPS. This write method may be slower than buffered writing, but no log information is lost if the system fails.

Example

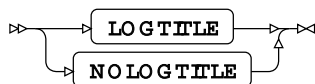
In this example, the system option is specified on the command line.

```
wps -log "c:\temp\temp.log" -logparm "open=append write=buffered" noo.wps
```

This directs WPS to write the log output to the file `c:\temp\temp.log`. If the log already exists, log output is appended to it. The output for the log is buffered and written to the log when the program finishes.

LOGTITLE

Specifies whether a title line is displayed at top of each page in the log output.



Valid in:	Configuration file and command line.
Default:	NOLOGTITLE
Option group:	LOGCONTROL
Portable	True
Restrictable	False
Saveable	False

LOGTITLE

Displays a title line at top of the log output, and at the start of each page.

NOLOGTITLE

No title line is displayed in the log output.

Example

In this example, the system option is specified on the command line.

```
wps -logtitle c:\temp\noo.wps
```

This directs WPS to write a title at the start of the log. The first few lines of the log look like this:

```
1                                     The WPS System   18:27 Thursday, February
  21, 2019

NOTE: (c) Copyright World Programming Limited 2002-2019. All rights reserved.
NOTE: World Programming System 4.01 (04.01.00.00.014673)
      Licensed to World Programming Company Ltd, 80 installations
NOTE: This session is executing on the X64_WIN10PRO platform and is running in 64
  bit mode

1      DATA _NULL_;
2      result1 = SCAN('london,,,bike,company;;A1', 3, ',;');
3      PUT result1=;
4      RUN;

:
```

The first line in the log is the title.

MEMRPT

Specifies whether to write memory usage statistics to the log when WPS starts.



Valid in: **OPTIONS** statement, configuration file and command line.

Option group: LOGCONTROL
 MEMORY

Portable False

Restrictable True

Saveable True

Supported platform: z/OS for System z

MEMRPT

Write memory usage statistics.

NOMEMRPT

Do not write memory usage statistics.

The statistics are written to the log before any text specified by the **NEWS** system option.

Example

In this example, the `OPTIONS` statement is used to specify that memory usage statistics are not written to the log.

```
OPTIONS NOMEMRPT;
```

MISSING

Specifies the character used to represent missing numeric values.

➤ **MISSING** ➤ = ➤ *character* ➤

Valid in: `OPTIONS` statement, configuration file and command line.

Default: .

Maximum length: 1

Option group: LOGCONTROL
LISTCONTROL
LOG_LISTCONTROL

Portable True

Restrictable True

Saveable True

The character specified is used to represent missing numeric values in logs, listing and datasets.

character

The character to use.

Example

In this example, the `OPTIONS` statement is used to specify that the + (plus) is used as the missing value character.

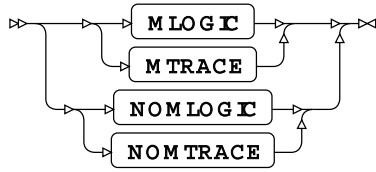
```
OPTIONS MISSING = '+';  
DATA _NULL_;  
  PUT x=;  
RUN;
```

This produces the following output:

```
NOTE: Variable "x" may not be initialized  
  
x=+
```

MLOGIC

Specifies whether to show in the log the execution of macros.



Valid in: `OPTIONS` statement, configuration file and command line.

Option group: `LOGCONTROL`
 `MACRO`

Portable True

Restrictable True

Saveable True

If this system option is set, the log shows when macro execution starts and stops.

MLOGIC

Log the execution of macros.

NOMLOGIC

Do not log the execution of macros.

If you also want to log executed macro statements, specify `MPRINT` [↗](#) (page 248). If you want log the execution of nested macros, also specify `MLOGICNEST` [↗](#) (page 247). If you want to log execute macro statements in nested macros displayed in the log, also specify both `MPRINT` and `MLOGICNEST`.

Example

In this example, the `OPTIONS` statement is used to specify that macro execution is displayed in log. The example shows the execution of a `DATA` step, and the result of the system option:

```
%MACRO test1;
  PUT "A test";
%MEND test1;

OPTIONS MLOGIC;
DATA _NULL_;

  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";

  %test1;

RUN;
```

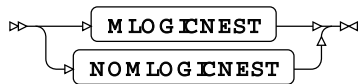
This produces the following output:

```
117      %MACRO test1;
118          PUT "A test";
119      %MEND test1;
120
121      OPTIONS MLOGIC;
122      DATA _NULL_;
123
124          b = 2 + 8;
125          PUT "This test program outputs " b "and then starts a macro";
126
127      %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1): Ending execution
128
129      RUN;
```

This test program outputs 10 and then starts a macro
A test
NOTE: The data step took :
 real time : 0.019
 cpu time : 0.015

MLOGICNEST

Specifies whether to log the execution of nested macros in MLOGIC output.



Valid in: OPTIONS statement, configuration file and command line.

Option group: LOGCONTROL
 MACRO

Portable True

MLOGICNEST

Log the execution of nested macros.

NOMLOGICNEST

Do not log the display of nested macros.

You must specify MLOGIC [↗](#) (page 246) if you use this option, otherwise no information on macro execution is written to the log.

If you also want to log the macro statements that were executed, specify MPRINT [↗](#) (page 248).

Example

In this example, the `OPTIONS` statement is used to specify that macro nesting information is written to the log. The following macros are created and then executed from a `DATA` step, the execution of the macros, and their nesting, is shown in the log.

```
%MACRO test;
  PUT "Output from a nested macro";
%MEND test;

%MACRO test1;
  %test;
%MEND test1;

OPTIONS MLOGIC MLOGICNEST;
DATA _NULL_;
  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";
  %test1;
RUN;
```

The following fragment from the log shows the execution of the `DATA` step and the result of the system options:

```
164      %MACRO test;
165          PUT "Output from a nested macro";
166      %MEND test;
167
168      %MACRO test1;
169          %test;
170      %MEND test1;
171
172      OPTIONS MLOGIC MLOGICNEST;
173      DATA _NULL_;
174          b = 2 + 8;
175          PUT "This test program outputs " b "and then starts a macro";
176          %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1.TEST): Beginning execution
MLOGIC(TEST1.TEST): Ending execution
MLOGIC(TEST1): Ending execution
177      RUN;

This test program outputs 10 and then starts a macro
Output from a nested macro
```

MPRINT

Specifies whether to display WPS statements generated by macro execution in the log.



Valid in:	OPTIONS statement, configuration file and command line.
Option group:	LOGCONTROL MACRO
Portable	True
Restrictable	True
Saveable	True

MPRINT

Display WPS statements generated by macro execution.

NOMPRINT

Do not display WPS statements generated by macro execution.

If you want to trace the execution of macros, also specify `MLOGIC` [↗](#) (page 246). If you want to see executed macro statements in nested macros, also specify `MPRINT` and `MLOGICNEST` [↗](#) (page 247).

Example

In this example, the `OPTIONS` statement is used to specify that macro execution is traced and statements in the macros are written to the log, and that WPS statements generated by macro execution are written to the log:

```
OPTIONS MLOGIC MLOGICNEST MPRINT;
```

The following fragment from the log shows the execution of the `DATA` step and the effect of the system options:

```
%MACRO test;
  PUT "Output from a nested macro";
%MEND test;

%MACRO test1;
  %test;
%MEND test1;

OPTIONS MLOGIC MLOGICNEST MPRINT;
DATA _NULL_;
  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";
  %test1;
RUN;
```

The following fragment from the log shows the execution of the `DATA` step, and the result of the system options:

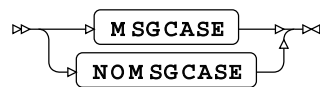
```
8      %MACRO test;
9      PUT "Output from a nested macro";
10     %MEND test;
11
12     %MACRO test1;
13     %test;
14     %MEND test1;
15
```

```
16      OPTIONS MLOGIC MLOGICNEST MPRINT;
17      DATA _NULL_;
18      b = 2 + 8;
19      PUT "This test program outputs " b "and then starts a macro";
20      %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1.TEST): Beginning execution
MPRINT(TEST): PUT "Output from a nested macro";
MLOGIC(TEST1.TEST): Ending execution
MPRINT(TEST1): ;
MLOGIC(TEST1): Ending execution
21      RUN;
```

This test program outputs 10 and then starts a macro
Output from a nested macro

MSGCASE

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOMSGCASE
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

MSGCASE

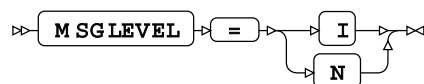
For compatibility only.

NOMSGCASE

For compatibility only.

MSGLEVEL

Specifies the amount of detail in returned messages.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	N
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

I

Return more detailed messages for dataset activities, indexes, and some procedures.

N

Return the normal level of detail.

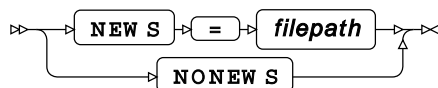
Example

In this example, the `OPTIONS` statement is used to specify that more information is returned.

```
OPTIONS MSGLEVEL=I
```

NEWS

Specifies the name of a file that contains messages to be written at the top of the log output.



Valid in:	Configuration file and command line.
Default:	<empty-string>
Maximum length:	255
Option group:	ENVFILES LOGCONTROL
Portable	True
Restrictable	True
Saveable	False

NEWS

Messages from the specified *filename* are written to the log.

NONEWS

News messages are not written to the log.

The messages are written at the top of the log, after information about WPS Analytics.

Example

In this example, the `NEWS` system option is specified on the command line. The file specified by the system option contains a message. The results are written to the log.

```
wps c:\temp\sadd.wps -news c:\temp\logmsg.txt
```

The file `logmsg.txt` contains the following:

```
Good day.  
Remember to lock your screen when you leave your desk.
```

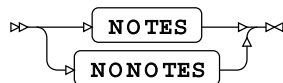
This produces the following output:

```
NOTE: (c) Copyright World Programming Limited 2002-2019. All rights reserved.  
NOTE: World Programming System 4.01 (04.01.00.00.014673)  
      Licensed to World Programming Company Ltd, 80 installations  
NOTE: This session is executing on the X64_WIN10PRO platform and is running in 64  
      bit mode  
  
Good day.  
Remember to lock your screen when you leave your desk.
```

Subsequent lines of the log contain program execution information, as usual.

NOTES

Specifies whether to display notes in the log output.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NONOTES
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

NOTES

Display notes in the log output.

NONOTES

Do not display notes in the log output.

If `NONOTES` is specified, only the program listing and any results written to the log are displayed in the log. Notes supply additional information, and are prefixed with `NOTE: .` Notes include information about WPS itself (version and the number of licences), and additional text that helps you understand log outputs.

Example

In this example, the system option is specified on the command line, and disables notes.

```
wps c:\temp\test2.wps -nonotes
```

The file `test2.wps` is a short program that outputs a message:

```
DATA _NULL_;
  PUT 'Hello there!';
RUN;
```

The resulting log contains the following:

```
1                                     The WPS System      10:55 Friday, June  1,
  2018

1          DATA _NULL_;
2          PUT 'Hello there!';
3          RUN;
Hello there!

          real time : 0.124
          cpu time  : 0.093
```

You can amend the command line to specify that notes are included in the log:

```
wps c:\temp\test2.wps -notes
```

The resulting log contains the following:

```
1                                     The WPS System      10:22 Friday, June  1,
  2018

NOTE: (c) Copyright World Programming Limited 2002-2018.  All rights reserved.
NOTE: World Programming System 4.01 (04.01.00.00.006862)
      Licensed to World Programming Company Ltd, 80 installations
NOTE: This session is executing on the X64_WIN10PRO platform and is running in 64
      bit mode

1          DATA _NULL_;
2          PUT 'This program runs';
3          RUN;

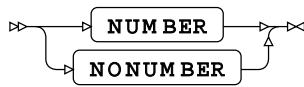
This program runs
NOTE: The data step took :
      real time : 0.000
      cpu time  : 0.000

NOTE: Submitted statements took :
      real time : 0.124
```

cpu time : 0.140

NUMBER

Specifies whether to print the page number at the top of each log and listing page.



Valid in: OPTIONS statement, configuration file and command line.

Default: NONUMBER

Option group: LOGCONTROL
LISTCONTROL
LOG_LISTCONTROL

Portable True

Restrictable True

Saveable True

NUMBER

Print the page number at the top of each page.

NONUMBER

Do not print the page number at the top of each page.

Example

In this example, the `OPTIONS` statement is used to specify that page numbers are not printed at the top of each page.

```

OPTIONS NONUMBER;
LIBNAME books 'C:\temp\books';
PROC REPORT DATA=books.lib_books;
WHERE Dewey_Decimal_Number GT "800" AND dewey_decimal_number LT "900";
column title;
BY author;
  
```

This writes the following to the first page of the log. There is no page number in the header at the top of the page.

```

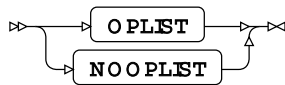
                                The WPS System
                                15:33 Wednesday, February 27, 2019
----- Author=Abraham, J H -----
Title
Origins and Growth of Sociology, The
  
```

If the option had been set to `NUMBER`, the page header would contain a page number:

The WPS System	1
15:33 Wednesday, February 27, 2019	
----- Author=Abraham, J H -----	
Title	
Origins and Growth of Sociology, The	1

OPLIST

Specifies whether to list WPS command line options in the log.



Valid in:	Configuration file and command line.
Default:	NOOPLIST
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	False

OPLIST

List command line options.

NOOPLIST

Do not list command line options.

Example

In this example, the system option is specified on the command line, and causes options to be listed to the log.

```
wps c:\temp\test2.wps -log -oplist
```

The log contains the following:

```

1                               The WPS System           11:13 Friday, June 1,
2018

NOTE: (c) Copyright World Programming Limited 2002-2018. All rights reserved.
NOTE: World Programming System 4.01 (04.01.00.00.006862)
      Licensed to World Programming Company Ltd, 80 installations
NOTE: This session is executing on the X64_WIN10PRO platform and is running in 64
      bit mode

NOTE: System options specified on command line are:
      LOG=c:\temp\op.log
  
```

```
OPLIST
SYSIN=c:\temp\test2.wps

1      DATA _NULL_;
2      PUT 'Hello there!';
3      RUN;

Hello there!
NOTE: The data step took :
      real time : 0.000
      cpu time  : 0.000

NOTE: Submitted statements took :
      real time : 0.140
      cpu time  : 0.140
```

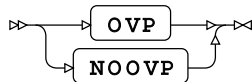
The list of options is introduced by the line:

```
NOTE: System options specified on command line are:
```

`SYSIN` is listed as one of the command line options. Although the program to execute it not specified using `SYSIN`, the system option `SYSIN` is implicitly inferred from the supplied WPS program name.

OVP

Specifies whether to overprint errors and warnings for emphasis.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOOVP
Option group:	LOGCONTROL
Portable	True
Restrictable	False
Saveable	True

OVP

Overprint errors and warnings.

NOOVP

Does not overprint errors and warnings.

Overprinting can be displayed or written on systems that support ANSI print control.

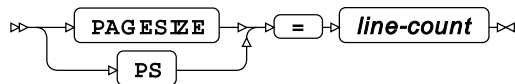
Example

In this example, the `OPTIONS` statement is used to specify that log information is emphasised.

```
OPTIONS OVP;
```

PAGESIZE

Specifies the number of lines that make up a page of logs and listings.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	55
Minimum value:	15
Maximum value:	32767
Option group:	LOGCONTROL LISTCONTROL LOG_LISTCONTROL
Portable	True
Restrictable	False
Saveable	True

line-count

The number of lines per page. This can be specified as:

- The number of lines; for example, if you enter 25, a page consists of 25 lines.
- The number of lines as a multiple of 1,024, by appending `K` to the value; for example, if the value is 0.2K, a page consists of 205 lines. Decimal values are rounded.
- The number of lines specified in hexadecimal, by appending `x`. For example, if the value is 1EX, a page consists of 30 lines.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

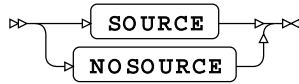
Example

In this example, the `OPTIONS` statement is used to specify a page length of 30 lines.

```
OPTIONS PAGESIZE = 30;
```

SOURCE

Specifies whether to list source statements in the log.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSOURCE
Option group:	LOGCONTROL
Portable	True
Restrictable	False
Saveable	True

SOURCE

List source statements in the log.

NOSOURCE

Do not list source statements in the log.

If you specify **NOSOURCE**, only the results of programs are listed in the log. No language statements (including those set by default by WPS, by Workbench, or by system options) are written to the log. The source of included programs is also listed. Included programs are those programs specified using **%INCLUDE**. To not list included programs, specify **NOSOURCE2** [↗](#) (page 259).

Example

In this example, the **OPTIONS** statement is used to specify that no source is listed in the log:

```
OPTIONS NOSOURCE;  
DATA _NULL_;  
  PUT 'Hello there!';  
RUN;
```

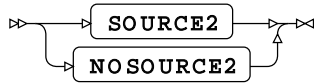
Only the following is written in the log:

```
Hello there!
```

No other information, such as source code, notes, and automatically generated code is listed.

SOURCE2

Specifies whether to show the content of included programs in the log output.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSOURCE2
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

SOURCE2

Show the content included programs.

NOSOURCE2

Do not show the content included programs.

Included programs are those programs specified using %INCLUDE. To prevent listing the content of included programs, specify NOSOURCE2 [↗](#) (page 259).

Example

In this example, the OPTIONS statement is used to specify that no included program content is included in the log:

```
OPTIONS SOURCE2;
DATA _NULL_;
  PUT 'Hello there!';
  %INCLUDE "c:\temp\noo.wps";
RUN;
```

The following is written in the log:

```
10      OPTIONS SOURCE2;
11      DATA _NULL_;
12
13          PUT 'Hello there!';
14          %INCLUDE "c:\temp\noo.wps";

Hello there!
NOTE: The data step took :
      real time : 0.000
      cpu time  : 0.000

Start of %INCLUDE(level 1) c:\temp\noo.wps
15      +
16      + DATA _NULL_;
```

```

17      +      num = 1 + 2 * 3 - 6;
18      +      PUT "Value of num is = " num;
19      +      RUN;

Value of num is = 1
NOTE: The data step took :
      real time : 0.000
      cpu time  : 0.000

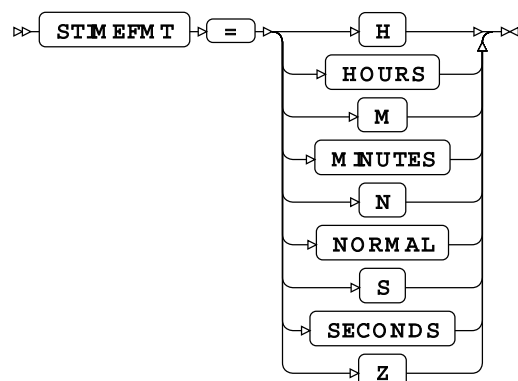
20      +
End of %INCLUDE(level 1) c:\temp\noo.wps
21
22      RUN;
23      quit; run;
24      ODS _ALL_ CLOSE;

```

Statements from the included program are included in the log, starting at line 15. If NOSOURCE2 is specified, only the value returned from the program (Value of num is = 1) would be written to the log.

STIMEFMT

Specifies the format of step timings in the log.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	N
Option group:	LOGCONTROL
Portable	True
Restrictable	True
Saveable	True

H

Display step timings as *hh:mm:ss.sss*.

HOURS

Display step timings as *hh:mm:ss.sss*.

M

Display step timings as *mm:ss.sss*.

MINUTES

Display step timings as *mm:ss.sss*.

N

The step timings display format is determined by the step duration:

- *ss.sss* if the time is less than one minute.
- *mm:ss.sss* if the time is greater than or equal to one minute, and less than one hour.
- *hh:mm:ss.sss* if the time is greater than or equal to one hour.

NORMAL

The step timings display format is determined by the step duration:

- *ss.sss* if the time is less than one minute.
- *mm:ss.sss* if the time is greater than or equal to one minute, and less than one hour.
- *hh:mm:ss.sss* if the time is greater than or equal to one hour.

S

Display step timings as *ss.sss*.

SECONDS

Display step timings as *ss.sss*.

Z

Display step timings as *hh:mm:ss.sss*.

For each format, *hh* is hours, *mm* is minutes, and *ss.sss* is seconds, including thousandth seconds.

Example

In this example, the `OPTIONS` statement is used to specify the format of step timings.

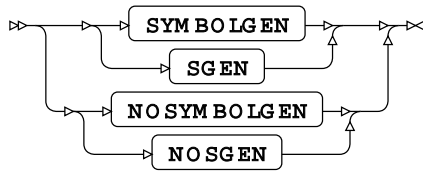
```
OPTIONS STIMEFMT=Z;  
DATA _NULL_;  
  DO i = 1 TO 1000;  
    DO x=1 TO 1000;  
      END;  
    END;  
  END;  
RUN;
```

The following is output in the log:

```
NOTE: The data step took :  
      real time : 0:00:00.003  
      cpu time  : 0:00:00.015
```

SYMBOLGEN

Specifies whether to write the results of resolved macro variable references to the log.



Valid in: `OPTIONS` statement, configuration file and command line.

Option group: `LOGCONTROL`
 `MACRO`

Portable True

Restrictable True

Saveable True

SYMBOLGEN

Write the results to the log.

NOSYMBOLGEN

Do not write the results to the log.

Example

In this example, the `OPTIONS` statement is used to write resolved macro variables to the log.

```
OPTIONS SYMBOLGEN;
```

You can then run the following program:

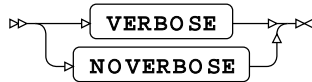
```
DATA _NULL_;  
  %LET xx = hello there;  
  to = CAT("Well ", "&xx");  
  PUT to;  
RUN;
```

This produces the following output:

```
SYMBOLGEN: Macro variable xx resolved to hello there  
  
Well hello there
```

VERBOSE

Specifies whether to list the value of configuration options in the log.



Valid in:	Configuration file and command line.
Default:	NOVERBOSE
Option group:	LOGCONTROL
Portable	False
Restrictable	True
Saveable	False

VERBOSE

List the value of configuration options.

NOVERBOSE

Do not list the value of configuration options.

The configuration options listed are the configuration files used when WPS starts, and the system options set. For system options, the option name and its value are listed.

Example

In this example, the system option is specified on the command line.

```
WPS c:\temp\testx.wps -VERBOSE
```

The following fragment from the log shows the format of the information listed:

```
==== Processed Configuration File(s) ====
c:\Program Files\World Programming\WPS\4\wps.cfg

Option      Value
=====
BOTTOMMARGIN 1CM
BYLINE      ON
CENTER       ON
CHARCODE     OFF
CONFIG       ( 'c:\Program Files\World Programming\WPS\4\wps.cfg' )
DATE         ON
DATESTYLE    MDY
DFLANG       ENGLISH
```

Note:

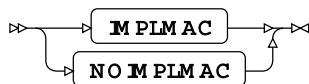
The configuration file shown is the default configuration file, as no file was specified on the command line.

MACRO group system options

System options that control macros including how they operate, the memory available to them, log output, and so on.

IMPLMAC

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOIMPLMAC
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

IMPLMAC

For compatibility only.

NOIMPLMAC

For compatibility only.

You can only specify `NOIMPLMAC`. If you specify `IMPLMAC`, a note is written to the log.

MACRO

Specifies whether the macro facility in WPS can be used.



Valid in:	Configuration file and command line.
Default:	NOMACRO
Option group:	MACRO
Portable	True
Restrictable	True

Saveable False

MACRO

The macro facility can be used.

NOMACRO

The macro facility cannot be used.

If **NOMACRO** is specified, and a program contains macro statements, functions or variables, these are not recognised and an error occurs.

Example

In this example, the system option is specified on the command line.

```
wps c:\temp\sadd.wps -nomacro
```

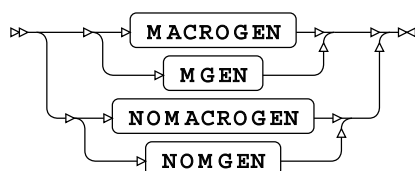
The file `sadd.wps` contains macro statements. These are not recognised as statements, and error messages are written to the log:

```
ERROR: Found "%" when expecting a statement
ERROR: The statement "LET" is unknown in this context
```

The program then fails to execute because errors have occurred.

MACROGEN

Specifies whether the execution of old-style macros is traced.



Valid in: OPTIONS statement, configuration file and command line.

Default: NOMACROGEN

Option group: MACRO

Portable True

Restrictable False

Saveable False

MACROGEN

Trace the execution of old-style macros.

NOMACROGEN

Do not trace the execution of old-style macros.

Example

In this example, the `OPTIONS` statement is used to specify that the execution is traced.

```
OPTIONS MACROGEN
```

If you then run the following statements, the log would contain information on where the old-style macro was executed.

```
MACRO test vv %
OPTIONS MACROGEN;
DATA _NULL_;
    test = 1;
    put vv;
RUN;
```

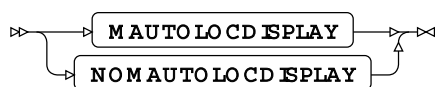
The log contains:

```
macro test vv %
793      options macrogen;
794      data _null_;
795      test = 1;
NOTE: The old-style macro test is beginning resolution
796      +    vv
NOTE: The old-style macro test is ending resolution
797      put vv;
798      run;
```

The lines identified by `NOTE` mark the trace of the old-style macro.

MAUTOLOCDISPLAY

Specifies whether to display the location of the autocall library.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOMAUTOLOCDISPLAY</code>
Option group:	<code>MACRO</code>
Portable	True
Restrictable	True
Saveable	True

MAUTOLOCDISPLAY

Display the location.

NOMAUTOLOCDISPLAY

Do not display the location.

The autocall library contains stored macros. The location of the autocall library is written to the log when a macro from the autocall library is run. The following message is written:

```
MAUTOLOCDISPLAY(macro): This macro was compiled from the autocall file filename
```

macro is the name of the macro. *filename* is the name of the file, including the path, that contains the macro.

The macros in the autocall library can only be accessed if you specify that autocall macros can be used with the MAUTOSOURCE [↗](#) (page 267) system option, and the location of the library has been specified using the SASAUTOS [↗](#) (page 103) system option.

Example

In this example, the OPTIONS statement is used to list the location of the autocall library.

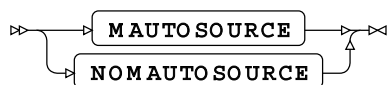
```
FILENAME msf "C:\temp\macros";  
options MAUTOSOURCE SASAUTOS=msf MAUTOLOCDISPLAY;
```

In this example, the specified folder contains a file named `test.wps` that contains a macro named `test`. If the macro is run, the following is written in the log:

```
MAUTOLOCDISPLAY(TEST): This macro was compiled from the autocall file C:\temp\macros  
\test.wps  
"Output from a macro"
```

MAUTOSOURCE

Enables the macro autocall facility.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOMAUTOSOURCE
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

MAUTOSOURCE

Enable the facility.

NOMAUTOSOURCE

Disable the facility.

The autocall library contains stored macros. The macros in the autocall library can only be accessed if you specify this system option. The location of the autocall library is specified using the [SASAUTOS](#) (page 103) system option.

Example

In this example, the `OPTIONS` statement is used to specify that the macro autocall facility is disabled.

```
OPTIONS NOMAUTOSOURCE
```

MCOMPILE

Specifies whether to compile macros.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOMCOMPILE`

Option group: `MACRO`

Portable: `True`

Restrictable: `True`

Saveable: `True`

MCOMPILE

Compile macros.

NOMCOMPILE

Do not compile macros.

If you specify `NOMCOMPILE`, macros cannot be compiled, and cannot be executed. However, this does not affect stored compiled macros that are read from a catalog, as these are already compiled.

Note:

If a macro has already been compiled and is in memory, subsequently setting `NOMCOMPILE` has no effect on that macro.

Example

In this example, the `OPTIONS` statement is used to specify that macros cannot be compiled.

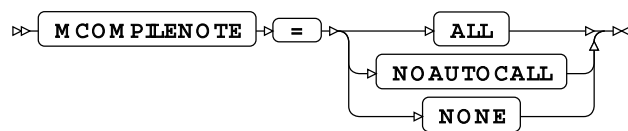
```
OPTIONS NOCOMPILE;
```

If you attempt to run a macro subsequent to setting this option, an error message is written to the log:

```
ERROR: The MCOMPILE option must be set in order to compile new macros
```

MCOMPILENOTE

Specifies that a note is written in the log if a macro is successfully compiled.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NONE
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

ALL

Add a note to the log for every macro that is compiled.

NOAUTOCALL

Add a note to the log for every macro that is compiled except for autocall macros.

NONE

Do not add a note to the log after compiling any macro.

The note has the format:

```
NOTE: The macro name completed compilation without errors  
      n instructions m bytes
```

Where *name* is the name of the macro, *n* is the number of instructions in the macro, and *m* is the number of bytes in the compiled macro.

Example

In this example, the `OPTIONS` statement is used to specify that a note is written to the log when a macro compiles successfully.

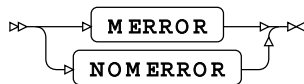
```
OPTIONS MCOMPILENOTE=ALL;  
%MACRO test2;  
DATA _NULL_;  
    PUT "Hello again";  
RUN;  
%MEND;
```

This returns a note in the log:

```
NOTE: The macro TEST2 completed compilation without errors  
      3 instructions  144 bytes
```

MERROR

Specifies whether to write a message to the log when an undefined macro reference occurs.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOMERROR</code>
Option group:	<code>MACRO</code>
Portable	True
Restrictable	True
Saveable	True

MERROR

Write a warning message.

NOMERROR

Do not write a warning message.

If a program references a macro that has not been compiled in the current session, the macro reference is not recognised and is treated as a syntax error. For example, if a program refers to an uncompiled macro `macro1`, the following error messages are written to the log:

```
ERROR: Found "%" when expecting a statement  
ERROR: The statement "macro1" is unknown in this context
```

It might, therefore, not be clear that an unresolved macro caused the error. If you set `MERROR`, an additional warning message is generated:

```
WARNING: Apparent invocation of macro "name" not resolved
```

name is the name specified for the macro.

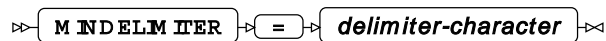
Example

In this example, the `OPTIONS` statement is used to specify a warning message is written for an unresolved macro reference.

```
OPTIONS MERROR;
```

MINDELIMITER

Specifies the character to use as the delimiter of the macro `IN` operator.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	1
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

delimiter-character

The character to use.

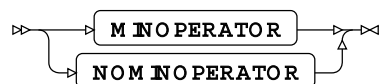
Example

In this example, the `OPTIONS` statement is used to specify the macro `IN` delimiter.

```
OPTIONS MINDELIMITER = "#";
```

MINOPERATOR

Specifies whether the `IN (#)` operator can be used in macros.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NOMINOPERATOR
Option group:	MACRO

Portable	True
Restrictable	True
Saveable	True

MINOPERATOR

The IN (#) operator can be used.

NOMINOPERATOR

The IN (#) operator cannot be used.

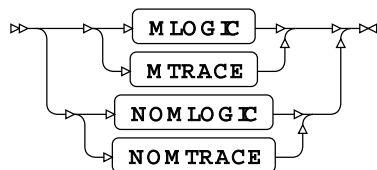
Example

In this example, the `OPTIONS` statement is used to specify that the `#` operator cannot be used.

```
OPTIONS NOMINOPERATOR;
```

MLOGIC

Specifies whether to show in the log the execution of macros.



Valid in: `OPTIONS` statement, configuration file and command line.

Option group: `LOGCONTROL`
`MACRO`

Portable	True
Restrictable	True
Saveable	True

If this system option is set, the log shows when macro execution starts and stops.

MLOGIC

Log the execution of macros.

NOMLOGIC

Do not log the execution of macros.

If you also want to log executed macro statements, specify `MPRINT` [↗](#) (page 248). If you want log the execution of nested macros, also specify `MLOGICNEST` [↗](#) (page 247). If you want to log execute macro statements in nested macros displayed in the log, also specify both `MPRINT` and `MLOGICNEST`.

Example

In this example, the `OPTIONS` statement is used to specify that macro execution is displayed in log. The example shows the execution of a `DATA` step, and the result of the system option:

```
%MACRO test1;
  PUT "A test";
%MEND test1;

OPTIONS MLOGIC;
DATA _NULL_;

  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";

  %test1;

RUN;
```

This produces the following output:

```
117      %MACRO test1;
118          PUT "A test";
119      %MEND test1;
120
121      OPTIONS MLOGIC;
122      DATA _NULL_;
123
124          b = 2 + 8;
125          PUT "This test program outputs " b "and then starts a macro";
126
127          %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1): Ending execution
128
129      RUN;

This test program outputs 10 and then starts a macro
A test
NOTE: The data step took :
      real time : 0.019
      cpu time  : 0.015
```

MLOGICNEST

Specifies whether to log the execution of nested macros in `MLOGIC` output.



Valid in: `OPTIONS` statement, configuration file and command line.

Option group: `LOGCONTROL`
`MACRO`

Portable True

MLOGICNEST

Log the execution of nested macros.

NOMLOGICNEST

Do not log the display of nested macros.

You must specify `MLOGIC` [↗](#) (page 246) if you use this option, otherwise no information on macro execution is written to the log.

If you also want to log the macro statements that were executed, specify `MPRINT` [↗](#) (page 248).

Example

In this example, the `OPTIONS` statement is used to specify that macro nesting information is written to the log. The following macros are created and then executed from a `DATA` step, the execution of the macros, and their nesting, is shown in the log.

```
%MACRO test;
  PUT "Output from a nested macro";
%MEND test;

%MACRO test1;
  %test;
%MEND test1;

OPTIONS MLOGIC MLOGICNEST;
DATA _NULL_;
  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";
  %test1;
RUN;
```


The following fragment from the log shows the execution of the `DATA` step and the result of the system options:

```
164      %MACRO test;
165          PUT "Output from a nested macro";
166      %MEND test;
167
168      %MACRO test1;
169          %test;
170      %MEND test1;
171
172      OPTIONS MLOGIC MLOGICNEST;
173      DATA _NULL_;
174          b = 2 + 8;
175          PUT "This test program outputs " b "and then starts a macro";
176          %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1.TEST): Beginning execution
MLOGIC(TEST1.TEST): Ending execution
MLOGIC(TEST1): Ending execution
177      RUN;
```

```
This test program outputs 10 and then starts a macro
Output from a nested macro
```

MPRINT

Specifies whether to display WPS statements generated by macro execution in the log.



Valid in: `OPTIONS` statement, configuration file and command line.

Option group: `LOGCONTROL`
 `MACRO`

Portable True

Restrictable True

Saveable True

MPRINT

Display WPS statements generated by macro execution.

NOMPRINT

Do not display WPS statements generated by macro execution.

If you want to trace the execution of macros, also specify `MLOGIC` [↗](#) (page 246). If you want to see executed macro statements in nested macros, also specify `MPRINT` and `MLOGICNEST` [↗](#) (page 247).

Example

In this example, the `OPTIONS` statement is used to specify that macro execution is traced and statements in the macros are written to the log, and that WPS statements generated by macro execution are written to the log:

```
OPTIONS MLOGIC MLOGICNEST MPRINT;
```

The following fragment from the log shows the execution of the `DATA` step and the effect of the system options:

```
%MACRO test;
  PUT "Output from a nested macro";
%MEND test;

%MACRO test1;
  %test;
%MEND test1;

OPTIONS MLOGIC MLOGICNEST MPRINT;
DATA _NULL_;
  b = 2 + 8;
  PUT "This test program outputs " b "and then starts a macro";
  %test1;
RUN;
```

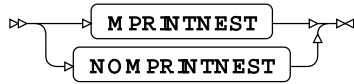
The following fragment from the log shows the execution of the `DATA` step, and the result of the system options:

```
8      %MACRO test;
9      PUT "Output from a nested macro";
10     %MEND test;
11
12     %MACRO test1;
13     %test;
14     %MEND test1;
15
16     OPTIONS MLOGIC MLOGICNEST MPRINT;
17     DATA _NULL_;
18     b = 2 + 8;
19     PUT "This test program outputs " b "and then starts a macro";
20     %test1;
MLOGIC(TEST1): Beginning execution
MLOGIC(TEST1.TEST): Beginning execution
MPRINT(TEST): PUT "Output from a nested macro";
MLOGIC(TEST1.TEST): Ending execution
MPRINT(TEST1): ;
MLOGIC(TEST1): Ending execution
21     RUN;
```

```
This test program outputs 10 and then starts a macro
Output from a nested macro
```

MPRINTNEST

Specifies whether to display macro nesting information in `MPRINT` output.



Valid in:	OPTIONS statement, configuration file and command line.
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

MPRINTNEST

Display macro nesting information.

NOMPRINTNEST

Do not display macro nesting information.

`MPRINT` must be specified to use this system option. When `MPRINT` is specified, each statement in a macro and in any nested macros, is written to the log. The name of the macro that contains the statement is appended to each line output to the log; for example:

```
MPRINT(TEST2): call symput("vc", "cheese");
```

If `MPRINTNEST` is specified and the macro is a nested macro, then the name of the macro that invoked it is appended to the nested macro's name; for example:

```
MPRINT(TEST1.TEST2): call symputx("vc", "cheese");
```

This example shows that the macro `TEST2` was executed by `TEST1`.

Example

In this example, the `OPTIONS` statement is used to specify that macro nesting information is written to the log. `MPRINT` is also specified, as `MPRINTNEST` modifies the output generated by it.

```
OPTIONS MPRINT MPRINTNEST
```

MRECALL

Specifies whether to search the autocall libraries for an undefined macro name each time it is invoked.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOMRECALL</code>
Option group:	<code>MACRO</code>
Portable	True
Restrictable	True
Saveable	True

MRECALL

Searches the autocall libraries for an undefined macro name.

NOMRECALL

Does not search the autocall libraries for an undefined macro name.

If an undefined macro name is specified in a program, then:

- If `NOMRECALL` is specified and the macro is not found, the autocall libraries are not searched again.
- If `MRECALL` is specified, and the macro is not found, the autocall libraries are searched again.

Example

In this example, the `OPTIONS` statement is used to specify that the autocall libraries are searched whenever an undefined macro name occurs.

```
OPTIONS MRECALL;
```

MSTORED

Specifies whether stored compiled macros can be used.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOMSTORED</code>
Option group:	<code>MACRO</code>
Portable	True
Restrictable	True
Saveable	True

MSTORED

Stored compiled macros can be used.

NOMSTORED

Stored compiled macros cannot be used.

Stored compiled macro are stored in catalogs. Macros can be compiled and stored in catalogs by using the `/STORE` option of the `MACRO` statement. To successfully create a stored compiled macro, `MSTORED` must be specified. If it is set to `NOMSTORED`, an error occurs, and the macro is not compiled. The following messages are written to the log:

```
ERROR: The MSTORED option must be set to use the /STORE macro statement option
macro statements
ERROR: The macro was not compiled
```

`MSTORED` must also be set to invoke a stored compiled macro. If it is set to `NOMSTORED`, the specified macro is not resolved, and an error occurs. The following messages are written to the log:

```
WARNING: Apparent invocation of macro "mname" not resolved
ERROR: Expected a statement keyword : found "%"
```

mname is the name of the macro.

To use a stored compiled macro, you also need to specify the location of the catalog that contains the macro. You do this using the `SASMSTORE` [🔗](#) (page 282) system option. Stored compiled macros are stored in a catalog called `SASMACR` in the specified location.

Example

In this example, the `OPTIONS` statement is used to specify that stored compiled macros can be used or generated.

```
OPTIONS MSTORED;
```

MSYMTABMAX

Specifies the maximum size of the macro variable symbol tables.

```
➤ M SYM TAB MAX ➤ = ➤ maximum-size ➤
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	4194304
Minimum value:	0
Maximum value:	2147483647
Option group:	<code>MACRO</code>
Portable	True
Restrictable	False
Saveable	True

Note:

The default above is for all operating systems except z/OS. The default on z/OS is 1,048,576 bytes.

maximum-size

The maximum table size, in bytes. This size can be specified as:

- The number of bytes; for example, you can enter 2147488.
- The number of kibibytes, mebibytes or gibibytes, by appending K, M or G to the value, respectively. For example, if the value is 2M, the buffer size is 2MiB.
- The number of bytes specified in hexadecimal, by appending X. For example, if the value is BEBC2X, the buffer size is 781250 bytes.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

The symbol tables comprise the area in memory in which macro variables are stored.

This system option enables you to control the memory allowed for the use of macro variable symbol tables.

If you attempt to create a macro variable once *maximum-size* is reached an error occurs.

Example

In this example, the `OPTIONS` statement is used to specify the maximum size of the symbol table.

```
OPTIONS MSYMTABMAX = 8388608;
```

MVARSIZE

Specifies the maximum size for in-memory macro variables.

```
➤ MVARSIZE ➤ = ➤ maximum-length ➤
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	4096
Minimum value:	0
Maximum value:	65534
Option group:	MACRO
Portable	True
Restrictable	False
Saveable	True

maximum-length

The maximum variable size, in bytes. This size can be specified as:

- The number of bytes; for example, you can enter 20000.
- The number of kibibytes, by appending **K** to the value, respectively. For example, if the value is 20K, the buffer size is 20KiB (20480 bytes). The number must be an integer.
- The number of bytes specified in hexadecimal, by appending **X**. For example, if the value is EBC2X, the buffer size is 60,354 bytes.
- **MIN** – the minimum supported value.
- **MAX** – the maximum supported value.

Example

In this example, the **OPTIONS** statement is used to specify the size of macro variables.

```
OPTIONS MVARSIZE = 1024
```

SASAUTOS

Specifies the list of locations to be searched for autocall macros.

```
>> SASAUTOS = >> library-specification <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	2048
Option group:	ENVFILES MACRO
Appendable	True
Portable	True
Restrictable	True
Saveable	True

This system option provides a list of locations to be searched for unknown macros encountered in SAS language programs. If an unknown macro is found in a source program, the specified locations are searched for source files with the name of that macro. If a suitable source file is found it is read and processed as if it had been included in the source program with the **%INCLUDE** statement.

You must also specify the **MAUTOSOURCE** [↗](#) (page 103) system option.

The source file does not have to contain a macro of the same name. The source file might instead contain only non-macro statements. In this case, a message is written to the log noting that the autocall member did not contain a macro of the same name. If you run the macro again, an error message is returned, unless you also set the **MRECALL** [↗](#) (page 277) system option.

library-specification

One or more locations to be searched. An autocall location can be specified as an operating system pathname, or as an existing filename reference, or as an external DD card on z/OS. If multiple locations are specified, enter them in parentheses. For example, to specify one autocall location `SASAUTOS = location-specification`; to specify two autocall locations, `SASAUTOS = (location-specification1 location-specification2)`

Enter an operating system pathname inside quotation marks.

The length of *library-specification* can only be 2048 bytes or less.

Example

In this example, the `OPTIONS` statement is used to specify that there are two autocall locations. The `SASAUTOS` option must be used with the `MAUTOSOURCE` system option.

```
FILENAME ms1 'c:\temp\macros';
OPTIONS MAUTOSOURCE SASAUTOS=(ms1 'c:\temp\macros2')
```

If one location contained the file `test.wps`, and the other the file `test2.wps`, you could compile and run the macros in these files by executing:

```
%test;
%test2;
```

SASMSTORE

Specifies the library location of the catalog that contains stored compiled macros.

```
➤ SASMSTORE = ➤ library-reference ➤
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	32
Option group:	MACRO
Portable	True
Restrictable	True
Saveable	True

library-reference

The path to the catalog, specified as a library name. You cannot specify an operating-system path name. The length of the path must be 32-bytes or less.

Compiled macros can be stored in a catalog. This catalog is called `SASMOCR`. It can be stored in any location. You can have more than one `SASMOCR`, stored in separate locations. You specify the location of the `SASMOCR` catalog you want to use through this system option. You specify the system option both when you create an entry in the catalog, and when you want to use a macro stored in the catalog. To create or use a stored macro, you also need to specify the `MSTORED` [↗](#) (page 278) system option.

A macro is stored in the catalog by setting the `/STORE` option of the `%MACRO` statement, the `MSTORED` system option, and the `SASMSTORE` system option to specify the location of the `SASMOCR` catalog.

A macro is read from the catalog by specifying the `MSTORED` system option, and this option to specify the location of the `SASMOCR` catalog.

Example

In this example, the `OPTIONS` statement is used to specify the location of the `SASMOCR` catalog. The `MSTORED` option is also specified, and a libname set.

```
LIBNAME ms "c:\temp\macros";  
OPTIONS MSTORED SASMSTORE=ms;
```

You can then run a compiled macro stored in the `SASMOCR` catalog in the specified library:

```
%test2
```

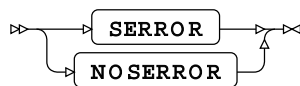
If the macro is not currently in memory, `SASMOCR` is searched for it.

You can compile and store a macro in `SASMOCR`, in the specified library:

```
%MACRO test3 /store;  
  %test2;  
  %PUT "Hello there!";  
%MEND test3;
```

SERROR

Specifies whether to generate a warning when an undefined macro variable reference occurs.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSERROR</code>
Option group:	<code>MACRO</code>
Portable	True
Restrictable	True
Saveable	True

SERROR

Write a warning message in the log.

NOSERROR

Do not write a warning message in the log.

If a program contains an undefined macro variable reference, by default the program runs to the end. The macro variable reference is treated as an unknown statement.

If **SERROR** is set, however, a warning message is written to the log:

```
WARNING: Macro variable "name" was not resolved
```

name is the name of the unresolved macro variable.

This can help you find problems in programs containing macro variables.

Example

In this example, the **OPTIONS** statement is used to specify that a warning is generated.

```
%MACRO test2;
DATA _NULL_;
    %put &var;
RUN;
%MEND;

OPTIONS SERROR;
%test2;
```

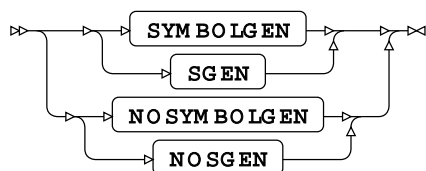
The following error message is written to the log:

```
WARNING: Macro variable "&var" was not resolved
```

If **NOSERROR** is set, the warning is written to the log.

SYMBOLGEN

Specifies whether to write the results of resolved macro variable references to the log.



Valid in: **OPTIONS** statement, configuration file and command line.

Option group: LOGCONTROL
MACRO

Portable: True

Restrictable	True
Saveable	True

SYMBOLGEN

Write the results to the log.

NOSYMBOLGEN

Do not write the results to the log.

Example

In this example, the `OPTIONS` statement is used to write resolved macro variables to the log.

```
OPTIONS SYMBOLGEN;
```

You can then run the following program:

```
DATA _NULL_;
  %LET xx = hello there;
  to = CAT("Well ", "&xx");
  PUT to;
RUN;
```

This produces the following output:

```
SYMBOLGEN: Macro variable xx resolved to hello there

Well hello there
```

MEMORY group system options

System options that control how memory is used by programs, and specifies sizes for memory use.

MEMRPT

Specifies whether to write memory usage statistics to the log when WPS starts.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Option group:	LOGCONTROL MEMORY
Portable	False

Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

MEMRPT

Write memory usage statistics.

NOMEMRPT

Do not write memory usage statistics.

The statistics are written to the log before any text specified by the `NEWS` system option.

Example

In this example, the `OPTIONS` statement is used to specify that memory usage statistics are not written to the log.

```
OPTIONS NOMEMRPT;
```

MEMSIZE

Specifies a limit on the total amount of memory used by the WPS.

➤ **MEMSIZE** ➤ = ➤ *memory-size* ➤

Valid in:	Configuration file and command line.
Default:	0
Minimum value:	0
Option group:	MEMORY
Portable	True
Restrictable	True
Saveable	False

memory-size

The total amount of memory, in bytes, used by WPS. This can be specified as:

- The number of bytes; for example, you can enter 2147483648.
- The number of kibibytes, mebibytes or gibibytes, by appending `K`, `M` or `G` to the value, respectively. For example, if the value is 2048M, the memory size is 2GiB.
- The number of bytes, specified in hexadecimal, by appending `x`. For example, if the value is 100011C0X, the memory size is 268440000 bytes.
- `MIN` – the minimum supported value.

- **MAX** – the maximum supported value.

If *memory-size* is set to 0 (zero), then is no constraint on the amount of memory WPS can use. The operating system on which WPS is running might have memory constraints already in place, in which case WPS uses the memory allowed within those constraints. If *memory-size* is set to some other value, then WPS uses the amount of memory specified; this might fail if other external constraints are already in place, in which case the amount allowed by the operating system is used.

Example

In this example, the system option is specified on the command line, and specifies that 1GiB of memory is available.

```
wps c:\temp\test2.wps -memsize 1G
```

MINSTG

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOMINSTG
Option group:	MEMORY
Portable	True
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

MINSTG

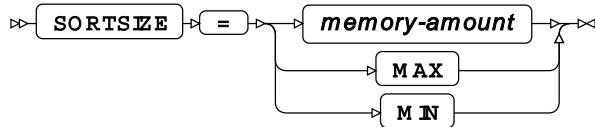
For compatibility only.

NOMINSTG

For compatibility only.

SORTSIZE

Specifies the amount of memory to use when performing a sort.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	MAX
Maximum length:	32
Option group:	SORT MEMORY PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

memory-amount

The total amount of memory used for the sort, in bytes. This can be specified as:

- The number of bytes; for example, you can enter 2147483648.
- The number of kibibytes, mebibytes or gibibytes, by appending K, M or G to the value, respectively. For example, if the value is 2048M, the memory available for sorting is 2GiB.
- The number, specified in hexadecimal, by appending x. For example, if the value is BEBC200X, the memory available for sorting is 200MB bytes.

The largest number you can enter is 32 characters long.

MAX

The maximum supported value.

MIN

The minimum supported value.

Example

In this example, the `OPTIONS` statement is used to specify that 100MiB of memory is available for a sort.

```
OPTIONS SORTSIZE = 100M;
```

SUMBUFNO

Specifies the size of the buffers to be used for summarising data.

➤ **SUMBUFNO** ➤ = ➤ *number-of-buffers* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	2147483647
Option group:	MEMORY
Portable	True
Restrictable	True
Saveable	True

Some procedures, such as `SUMMARY` or `TABULATE`, create summaries of data. This system option enables you specify the number of buffers used to store temporary data while creating summaries.

number-of-buffers

The number of buffers required. This can be specified as:

- A number; for example, you can enter 2147.
- A number of buffers multiplied by 1024, by 1024² or 1024³, by appending K, M or G to the value, respectively. For example, if the value is 1M, the number of buffers is the 1,048,576.
- A number specified in hexadecimal, by appending X. For example, if the value is EBC2X, the number of buffers is 60,354.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

If *number-of-buffers* is set to 0 (zero), then WPS uses 500 buffers; otherwise, it uses the number specified.

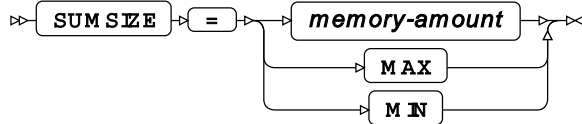
Example

In this example, the `OPTIONS` statement is used to specify 1024 buffers.

```
OPTIONS SORTBUFNO = 1K;
```

SUMSIZE

Specifies the maximum amount of memory available to the `SUMMARY` and `MEANS` procedures.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	MAX
Maximum length:	32
Option group:	MEMORY
Portable	True
Restrictable	True
Saveable	True

This system option specifies how much memory is available to summarisation operations in procedures that create summary data, such as `SUMMARY` and `MEANS`.

memory-amount

The maximum memory size available, in bytes. This can be specified as:

- The number of bytes; for example, you can enter 2147488.
- The number of kibibytes, mebibytes or gibibytes, by appending `K`, `M` or `G` to the value, respectively. For example, if the value is 2M, the amount of memory available is 2097152 bytes.
- The number of bytes specified in hexadecimal, by appending `X`. For example, if the value is BEBC200X, the memory available for sorting is 200MB.

The number specified can be no longer than 32 characters.

MAX

The maximum supported value.

MIN

The minimum supported value.

Example

In this example, the `OPTIONS` statement is used to specify that x'21A78F' of memory is available:

```
OPTIONS SUMSIZE = 21A78FX;
```


ODSPRINT group system options

System options that specify ODSPRINT settings.

BOTTOMMARGIN

Specifies the width of the bottom margin of PDFs created using the Output Delivery System (ODS).

➤ **BOTTOMMARGIN** ➤ = ➤ *margin-size* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0.0 in
Maximum length:	10
Option group:	ODSPRINT
Appendable	False
Portable	False
Restrictable	False
Saveable	False

margin-size

The margin size. The size can be expressed in inches (*in*), centimetres (*cm*) or points (*pt*). The size can be entered with or without quotes. For example, *1in*, *"2cm"*, *10pt*.

This option can be overridden using the equivalent statement on the ODS PDF statement.

Example

In this example, the OPTIONS statement is used to specify that the bottom margin of the PDF output is 1cm:

```
OPTIONS BOTTOMMARGIN = 1cm
```

LEFTMARGIN

Specifies the width of the left margin in PDFs created using the Output Delivery System (ODS).

➤ **LEFTMARGIN** ➤ = ➤ *margin-size* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0.0 in

Maximum length:	10
Option group:	ODSPRINT
Appendable	False
Portable	False
Restrictable	False
Saveable	False

margin-size

The margin width. The width can be expressed in inches (*in*), centimetres (*cm*) or points (*pt*).

The size can be entered with or without quotes. For example, *1in*, *"2cm"*, *10pt*.

This option can be overridden using the equivalent statement on the `ODS PDF` statement.

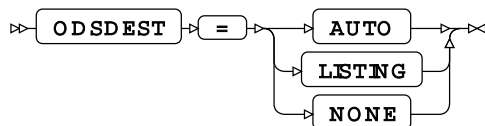
Example

In this example, the `OPTIONS` statement is used to specify that the left margin width of the PDF output is 1cm:

```
OPTIONS LEFTMARGIN = 1cm;
```

ODSDEST

Specifies the default ODS destination to initialise.



Valid in:	Configuration file and command line.
Default:	LISTING
Option group:	ODSPRINT
Portable	True
Restrictable	True
Saveable	True

AUTO

Create a listing file.

LISTING

Create a listing file.

NONE

No default ODS destination.

AUTO and LISTING currently function the same way. If NONE is specified, a destination needs to be specified in programs.

Example

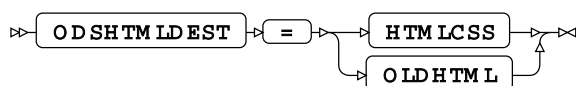
In this example, the option is specified on the command line.

```
wps sadd.wps -odsdest listing
```

Output from the program `sadd.wps` is written to the ODS listing destination.

ODSHTMLDEST

Specifies the type of HTML file created when writing HTML files using ODS HTML.



Valid in: Configuration file and command line.

Default: HTMLCSS

Option group: ODSPRINT

Portable: True

HTMLCSS

A file is created that is equivalent to specifying ODS HTMLCSS.

OLDHTML

A file is created that is equivalent to specifying ODS OLDHTML.

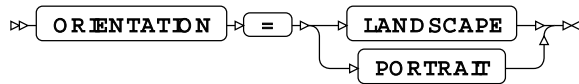
Example

In this example, the `OPTIONS` statement is used to specify that the type of HTML file generated is the same as that generated using ODS OLDHTML:

```
wps sadd -odshtmldest oldhtml;
```

ORIENTATION

Specifies the page orientation of the PDF created using ODS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	PORTRAIT
Option group:	ODSPRINT
Portable	True
Restrictable	True
Saveable	True

LANDSCAPE

Create PDF in landscape orientation.

PORTRAIT

Create PDF in portrait orientation.

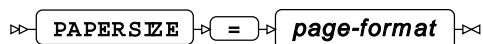
Example

In this example, the `OPTIONS` statement is used to specify the landscape orientation for PDF output:

```
OPTIONS ORIENTATION = LANDSCAPE;
```

PAPERSIZE

Specifies the paper size for PDFs created using ODS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	A4
Maximum length:	35
Option group:	ODSPRINT
Portable	True
Restrictable	False
Saveable	False

page-format

A string that specifies a corresponding, supported page size, and must be one of:

- A2
- A3
- A4
- A5
- LETTER
- USLETTER

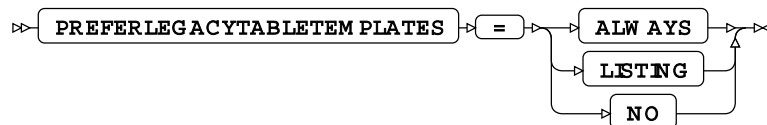
Example

In this example, the `OPTIONS` statement is used to specify the page size for PDF output:

```
OPTIONS PAGESIZE = A5;
```

PREFERLEGACYTABLETEMPLATES

Specifies whether version 4.1 table templates are used



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `LISTING`

Option group: `ODSPRINT`

Portable: `False`

Restrictable: `True`

Saveable: `False`

ALWAYS

Version 4.1 table templates are used for all ODS destinations.

LISTING

Version 4.1 tables templates are used for the ODS `LISTING` destination. All other destinations use the pre-version 4.1 table templates.

NO

Pre-Version 4.1 table templates are used for all ODS destinations.

Example

In this example, the `OPTIONS` statement is used to specify that version 4.1 tables templates are used for the ODS LISTING destination.

```
OPTIONS PREFERLEGACYTABLETEMPLATES=LISTING
```

RIGHTMARGIN

Specifies the right margin width of PDFs output via the Output Delivery System (ODS).

➤ **RIGHTMARGIN** ➤ = ➤ *margin-size* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	0.0 in
Maximum length:	10
Option group:	ODSPRINT
Appendable	False
Portable	False
Restrictable	False
Saveable	False

margin-size

The margin width. The width can be expressed in inches (`in`), centimetres (`cm`) or points (`pt`). The size can be entered with or without quotes. For example, `1in`, `"2cm"`, `10pt`.

Example

In this example, the `OPTIONS` statement is used to specify that the right margin of the PDF output is 1cm:

```
OPTIONS RIGHTMARGIN = 1cm;
```

TOPMARGIN

Specifies the width of the top margin of PDFs created using the Output Delivery System (ODS).

➤ **TOPMARGIN** ➤ = ➤ *margin-size* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	0.0 in

Maximum length:	10
Option group:	ODSPRINT
Appendable	False
Portable	False
Restrictable	False
Saveable	False

margin-size

The margin size. The size can be expressed in inches (*in*), centimetres (*cm*) or points (*pt*). The size can be entered with or without quotes. For example, *1in*, *"2cm"*, *10pt*.

This option can be overridden using the equivalent statement on the `ODS PDF` statement.

Example

In this example, the `OPTIONS` statement is used to specify that the top margin of the PDF output is 1cm:

```
OPTIONS TOPMARGIN = 1cm;
```

PERFORMANCE group system options

System options that affect program performance.

BUFNO

Specifies the number of buffers used for simple file operations, such as that provided by the `FILE` and `INFILE` statements.

➤ **BUFNO** ➤ **=** ➤ *number-of-buffers* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	0
Maximum value:	2147483647
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True

Saveable True

number-of-buffers

The number of buffers to use. This can be specified as:

- The number of buffers; for example, you can enter 2147488.
- The number of buffers as a multiple of 1,024, 1,024² or 1,024³, by appending K, M or G to the value, respectively. For example, if the value is 100K, the number of buffers is 102,400.
- The number of buffers specified in hexadecimal, by appending x. For example, if the value is BEBC2X, the number of buffers available is 781,250.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

Increasing the number of buffers used to read data from and write data to a file might increase the speed with which data can be read; however, reducing the amount of memory available for other operations might slow overall performance.

Example

In this example, the `OPTIONS` statement is used to specify that 32 buffers are available.

```
OPTIONS BUFNO = 32;
```

BUFSIZE

Specifies the default size of a page used when creating WPS datasets.

➤ **BUFSIZE** ➤ = ➤ *buffer-size* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	2147483647
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

This system option only affects the size of the page used when creating new WPS datasets, not when interacting with existing datasets.

buffer-size

The size of the page, in bytes. This size can be specified as:

- The number of bytes; for example, you can enter 2147488.
- The number of kibibytes, mebibytes or gibibytes, by appending **K**, **M** or **G** to the value, respectively. For example, if the value is 2M, the buffer size is 2MiB.
- The number of bytes, specified in hexadecimal, by appending **X**. For example, if the value is BEBC2X, the buffer size is 781250 bytes.
- **MIN** – the minimum supported value.
- **MAX** – the maximum supported value.

WPS rounds any numeric value specified to produce a useable value. If the value provided is too small for the dataset or is 0 (zero), WPS determines a value.

The buffer size can be tailored for compressed and uncompressed datasets by also specifying the **BUFSIZECMULT** and **BUFSIZEUMULT** system options, respectively.

Example

In this example, the **OPTIONS** statement is used to specify that the size of the buffer is 1MiB.

```
OPTIONS BUFSIZE = 1M;
```

BUFSIZECMULT

Specifies a multiplier that is applied to the computed minimum page size for a compressed WPS dataset.

```
BUFSIZECMULT = multiplier
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	6
Minimum value:	1
Maximum value:	65535
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

You can, in general, leave this at the default value. However, for some datasets, increasing the value might give better performance, but this will be at the cost of more memory.

multiplier

The multiplier to be applied.

The value specified here is a multiplier to the value specified to `BUFSIZE`. For a compressed dataset, therefore, the buffer size used is the result of the value of `BUFSIZE` multiplied by the value of `BUFSIZECMULT`.

Example

In this example, the `OPTIONS` statement is used to specify that the size of the minimum computed size of a buffer is multiplied by 5.

```
OPTIONS BUFSIZECMULT = 5;
```

BUFSIZEUMULT

Specifies a multiplier that is applied to the computed minimum page size for an uncompressed WPS dataset.

⇒ `BUFSIZEUMULT` ⇒ `=` ⇒ *multiplier* ⇒

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	1
Maximum value:	65535
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

You can, in general, leave this at the default value. However, for some datasets, increasing the value might give better performance, but this will be at the cost of more memory.

multiplier

The value specified here is a multiplier to the value specified to `BUFSIZE`. For an uncompressed dataset, therefore, the buffer size used is the result of the value of `BUFSIZE` multiplied by the value of `BUFSIZEUMULT`.

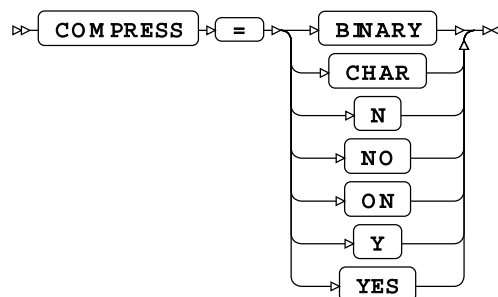
Example

In this example, the `OPTIONS` statement is used to specify that the size of the minimum computed size of a page is multiplied by 5.

```
OPTIONS BUFSIZEUMULT = 5;
```

COMPRESS

Specifies whether to compress the data when WPS datasets are created.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NO
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

Compression can be useful if storage space is limited; however, there is a processing cost to compressing and decompressing files. Similarly, it might be quicker to read and write compressed files across networks, but the files subsequently need to be decompressed. There are, then, trade-offs that must be considered before applying this system option.

The amount by which data can be compressed depends on the type of data to be compressed.

BINARY

Use Ross Data Compression (RDC) algorithm.

CHAR

Use run-length encoding (RLE) algorithm.

N

Do not compress.

NO

Do not compress.

ON

Use run-length encoding (RLE) algorithm.

Y

Use run-length encoding (RLE) algorithm.

YES

Use run-length encoding (RLE) algorithm.

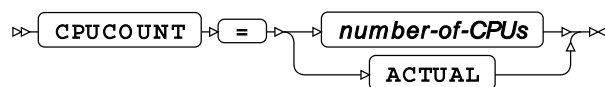
Example

In this example, the `OPTIONS` statement is used to that RDC is used to compress datasets.

```
OPTIONS COMPESS = BINARY;
```

CPUCOUNT

Specifies the number of CPUs available to programs.



Valid in: `OPTIONS` statement, configuration file and command line.

Minimum value: 1

Maximum value: 1024

Option group: `PERFORMANCE`

Portable: False

Restrictable: True

Saveable: False

number-of-CPU's

The number of physical or logical CPUs that are available to the application.

ACTUAL

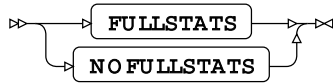
Example

In this example, the `OPTIONS` statement is used to specify that two CPUs can be used.

```
OPTIONS CPUCOUNT = 2;
```

FULLSTATS

Specifies whether to write more detailed performance statistics for a step on z/OS.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFULLSTATS`

Option group: `LOGCONTROL`
 `PERFORMANCE`

Supported platform: z/OS for System z

FULLSTATS

Write more detailed performance statistics.

NOFULLSTATS

Do not write more detailed performance statistics.

By default, the information returned is file information and the amount of real time and processor time used to execute the step. If you specify `FULLSTATS`, additional information is provided about the EXCP count. This information is added after the data step timing; for example:

```
real time : 0.002
cpu time  : 0.001
EXCP count: 0
```

Example

In this example, the `OPTIONS` statement is used to specify full statistics are written to the log.

```
OPTIONS FULLSTATS;
```

If a program is then run, the following is written to the log:

```
The file INLINE is:

Dsname = CJH.FULLSTAT.JOB08311.D0000101.?,
Unit =,
Volume =, Disp = NEW, Blksize=80, Lrecl=80, Recfm=FB
Creation=2018/08/01

LINE
1

LINE
2

NOTE: 2 records were read from file
      INLINE
NOTE: The data step took:

      real time : 0.042
      cpu time  : 0.003
      EXCP count: 0
```

The EXCP count has been added to the log.

IBUFNO

Specifies the number of index file buffers used by a library engine for a dataset.

```
<< IBUFNO = >> buffer-number <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	10000
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

buffer-number

The total amount of file index buffers available. This can be specified as:

- The number of buffers; for example, you can enter 5000.

- The number of buffers as a multiple of 1024, by appending **K** to the value. For example, if the value is **7K**, 7,168 index file buffers are used. This number must be an integer.
- The of buffers specified in hexadecimal, by appending **X**. For example, if the value is **1000X**, the number of buffers is 4096.
- **MIN** – the minimum supported value.
- **MAX** – the maximum supported value.

Example

In this example, the **OPTIONS** statement is used to specify that 5000 index file buffers are available.

```
OPTIONS IBUFNO = 5000;
```

IBUFSIZE

Specifies the size of an index page for a WPS dataset.

```
<> IBUFSIZE = <buffer-size>
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	32767
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

buffer-size

The size of index page to use, in bytes. This can be specified as:

- The number of bytes; for example, you can enter **5000**.
- The number of kibibytes, by appending **K** to the value. For example, if the value is **7K**, the index page size is 7,168 bytes. The number must be an integer.
- **MIN** – the minimum supported value.
- **MAX** – the maximum supported value.

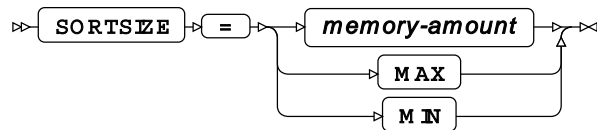
Example

In this example, the `OPTIONS` statement is used to specify that the index page is eight kibibytes in size.

```
OPTIONS IBUFNO = 8K;
```

SORTSIZE

Specifies the amount of memory to use when performing a sort.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>MAX</code>
Maximum length:	32
Option group:	<code>SORT</code> <code>MEMORY</code> <code>PERFORMANCE</code>
Portable	True
Restrictable	True
Saveable	True

memory-amount

The total amount of memory used for the sort, in bytes. This can be specified as:

- The number of bytes; for example, you can enter 2147483648.
- The number of kibibytes, mebibytes or gibibytes, by appending `K`, `M` or `G` to the value, respectively. For example, if the value is 2048M, the memory available for sorting is 2GiB.
- The number, specified in hexadecimal, by appending `x`. For example, if the value is BEBC200X, the memory available for sorting is 200MB bytes.

The largest number you can enter is 32 characters long.

MAX

The maximum supported value.

MIN

The minimum supported value.

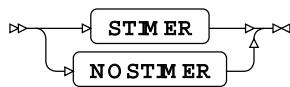
Example

In this example, the `OPTIONS` statement is used to specify that 100MiB of memory is available for a sort.

```
OPTIONS SORTSIZE = 100M;
```

STIMER

Specifies whether to write performance statistics to the log after each step.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOSTIMER`

Option group: `PERFORMANCE`
`SMF`

Portable: `False`

Restrictable: `True`

Saveable: `False`

STIMER

Write performance statistics after each step.

NOSTIMER

Do not write performance statistics after each step.

A step can be the execution of a procedure or `DATA` step.

Example

In this example, the `OPTIONS` statement is used to specify that performance statistics are not written to the log.

```
OPTIONS NOSTIMER;
```

THREADS

Specifies whether to enable multi-threaded processing.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOTHREADS
Option group:	PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

THREADS

Enable multi-threaded processing.

NOTHREADS

Disable multi-threaded processing.

You can specify `THREADS` on a system on which multi-threaded processing is not available; no error will occur.

Example

In this example, the `OPTIONS` statement is used to specify that multi-threaded processing is not available.

```
OPTIONS NOTHREADS;
```

WPDHUGE

Specifies whether new WPD datasets are permitted to have more than 2^{31} records.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOWPDHUGE
Option group:	PERFORMANCE
Portable	True

Restrictable True

Saveable True

WPDHUGE

Allow datasets to have more than 2^{31} records.

NOWPDHUGE

Do not allow datasets to have more than 2^{31} records.

Example

In this example, the `OPTIONS` statement is used to specify that datasets cannot have more than 2^{31} records.

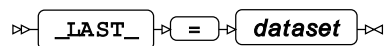
```
OPTIONS NOWPDHUGE;
```

SASFILES group system options

System options that effect datasets.

`__LAST__`

Sets the system variable `__LAST__` to the specified dataset name.



Valid in: `OPTIONS` statement, configuration file and command line.

Maximum length: 1024

Option group: SASFILES

Portable True

Restrictable False

Saveable True

dataset

Assigns a specific dataset to the system variable `__LAST__`.

The system variable `__LAST__` can be used in place of a dataset name:

```
PROC CONTENTS DATA=__LAST__;  
RUN;
```

The `CONTENTS` procedure then use the last dataset used in the session. However, you can specify a dataset name as shown in the example below, and then instead of the last dataset, the specified dataset is used.

Some procedures use the value of the system variable `_LAST_` by default if no dataset name is specified.

Example

In this example, the `OPTIONS` statement is used to specify that the system variable `_LAST_` contains a specified name.

```
OPTIONS _LAST_ = myDataset;
```

BASEENGINE

Specifies the library engine to use when the `BASE` option is specified in `LIBNAME`.

⇒ **BASEENGINE** ⇒ = ⇒ *engine-name* ⇒

Valid in:	Configuration file and command line.
Default:	WPD
Maximum length:	8
Option group:	SASFILES
Portable	True
Restrictable	False
Saveable	False

engine-name

The name of the library engine to use. This is one of the library engine names described in the section *Library engines*.

`BASE` is a pseudo-engine name that can be used in a `LIBNAME` statement. If `BASE` is specified, `BASEENGINE` must also be set.

Example

In this example, the option is specified on the command line. It specifies that the `BASE` pseudo-engine refers to the Access library engine.

```
wps c:\temp\test2.wps -baseengine access
```

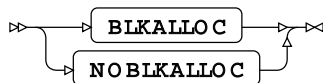
The program `test2.wps` is:

```
LIBNAME be BASE "c:\temp\test1.accdb";
DATA be.table1;
  INPUT txt;
  FORMAT txt 2.;
CARDS;
6
;
```

This writes the data to the table `table1` in the specified Access database.

BLKALLOC

Specifies whether a block size is defined when dynamically allocating storage space for a data library in an MVS dataset.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOBLKALLOC
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

BLKALLOC

The default block size, set using the `BLKSIZE` option, is always specified when allocating storage space for a new data library.

NOBLKALLOC

The default block size, set using the `BLKSIZE` option, can be zero when dynamically allocating storage space for a new data library.

The size of a block is normally set using the `BLKSIZE` option of a `LIBNAME` statement. If that option is not specified, then:

- If `BLKALLOC` is set, one of the following occurs:
 - The block size is set to the value specified by the `BLKSIZE` system option.
 - If no value is specified to the `BLKSIZE` system option, the block size is set to the value specified by the `BLKSIZE(OTHER)` system option.
 - If no value is specified for the `BLKSIZE` or `BLKSIZE(OTHER)` system options, the default block size is 6164 bytes.

- If NOBLKALLOC is set, WPS automatically chooses a block size.

Example

In this example, the `OPTIONS` statement is used to specify that the default block size for a new data library is set dynamically.

```
OPTIONS NOBLKALLOC;
```

BLKSIZE

Specifies the default block size for data libraries.

```
BLKSIZE = block-size
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	27998
Minimum value:	1024
Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that the block size for data libraries is 27998 bytes.

```
OPTIONS BLKSIZE = 27998;
```

BLKSIZE(3375)

Specifies the default block size for data libraries on 3375 devices.

```
BLKSIZE(3375) = block-size
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	8192
Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(3375) = 16384;
```

BLKSIZE(3380)

Specifies the default block size for data libraries on 3380 devices.

```
BLKSIZE(3380) = block-size
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	23476
Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(3380) = 16384;
```

BLKSIZE(3390)

Specifies the default block size for data libraries on 3390 devices.

➤ **BLKSIZE(3390)** ➤ **=** ➤ **block-size** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	27998
Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(3390) = 16384;
```

BLKSIZE(9345)

Specifies the default block size for data libraries on 9345 devices.

➤ **BLKSIZE(9345)** ➤ **=** ➤ **block-size** ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	6144

Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

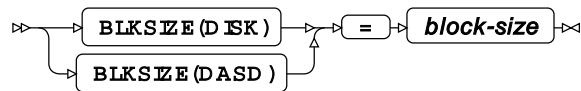
Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(9345) = 16384;
```

BLKSIZE(DISK)

Specifies the default block size for data libraries on DISK devices.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	0
Maximum value:	32760
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

If *block-size* is set to 0 (zero), WPS calculates a block size based on the size of observations in the dataset.

Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(DISK) = 16384;
```

BLKSIZE(OTHER)

Specifies the default block size for data libraries on `OTHER` devices.

➤ `BLKSIZE(OTHER)` ➤ `=` ➤ *block-size* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	6144
Maximum value:	32760
Option group:	<code>SASFILES</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The amount of storage space, in bytes, to allocate for each block.

Example

In this example, the `OPTIONS` statement is used to specify that the blocksize for data libraries is 16384 bytes.

```
OPTIONS BLKSIZE(OTHER) = 16384;
```

BUFNO

Specifies the number of buffers used for simple file operations, such as that provided by the `FILE` and `INFILE` statements.

➤ `BUFNO` ➤ `=` ➤ *number-of-buffers* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
-----------	--

Default:	1
Minimum value:	0
Maximum value:	2147483647
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

number-of-buffers

The number of buffers to use. This can be specified as:

- The number of buffers; for example, you can enter 2147488.
- The number of buffers as a multiple of 1,024, 1,024² or 1,024³, by appending K, M or G to the value, respectively. For example, if the value is 100K, the number of buffers is 102,400.
- The number of buffers specified in hexadecimal, by appending x. For example, if the value is BEBC2X, the number of buffers available is 781,250.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

Increasing the number of buffers used to read data from and write data to a file might increase the speed with which data can be read; however, reducing the amount of memory available for other operations might slow overall performance.

Example

In this example, the `OPTIONS` statement is used to specify that 32 buffers are available.

```
OPTIONS BUFNO = 32;
```

BUFSIZE

Specifies the default size of a page used when creating WPS datasets.

```
>> [BUFSIZE] [=] [buffer-size] <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	2147483647
Option group:	SASFILES

	PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

This system option only affects the size of the page used when creating new WPS datasets, not when interacting with existing datasets.

buffer-size

The size of the page, in bytes. This size can be specified as:

- The number of bytes; for example, you can enter 2147488.
- The number of kibibytes, mebibytes or gibibytes, by appending K, M or G to the value, respectively. For example, if the value is 2M, the buffer size is 2MiB.
- The number of bytes, specified in hexadecimal, by appending x. For example, if the value is BEBC2X, the buffer size is 781250 bytes.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

WPS rounds any numeric value specified to produce a useable value. If the value provided is too small for the dataset or is 0 (zero), WPS determines a value.

The buffer size can be tailored for compressed and uncompressed datasets by also specifying the BUFSIZECMULT and BUFSIZEUMULT system options, respectively.

Example

In this example, the `OPTIONS` statement is used to specify that the size of the buffer is 1MiB.

```
OPTIONS BUFSIZE = 1M;
```

BUFSIZECMULT

Specifies a multiplier that is applied to the computed minimum page size for a compressed WPS dataset.

```
➤ BUFSIZECMULT = multiplier ➤
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	6
Minimum value:	1
Maximum value:	65535

Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

You can, in general, leave this at the default value. However, for some datasets, increasing the value might give better performance, but this will be at the cost of more memory.

multiplier

The multiplier to be applied.

The value specified here is a multiplier to the value specified to `BUFSIZE`. For a compressed dataset, therefore, the buffer size used is the result of the value of `BUFSIZE` multiplied by the value of `BUFSIZEMULT`.

Example

In this example, the `OPTIONS` statement is used to specify that the size of the minimum computed size of a buffer is multiplied by 5.

```
OPTIONS BUFSIZEMULT = 5;
```

BUFSIZEMULT

Specifies a multiplier that is applied to the computed minimum page size for an uncompressed WPS dataset.

```
BUFSIZEMULT = multiplier
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	1
Maximum value:	65535
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

You can, in general, leave this at the default value. However, for some datasets, increasing the value might give better performance, but this will be at the cost of more memory.

multiplier

The value specified here is a multiplier to the value specified to `BUFSIZE`. For an uncompressed dataset, therefore, the buffer size used is the result of the value of `BUFSIZE` multiplied by the value of `BUFSIZEUMULT`.

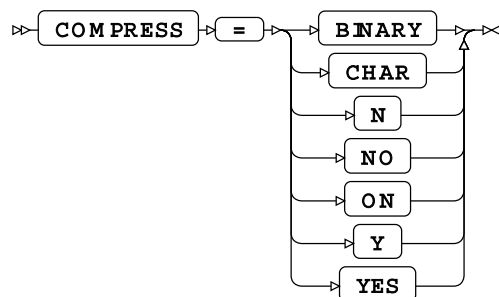
Example

In this example, the `OPTIONS` statement is used to specify that the size of the minimum computed size of a page is multiplied by 5.

```
OPTIONS BUFSIZEUMULT = 5;
```

COMPRESS

Specifies whether to compress the data when WPS datasets are created.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NO
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

Compression can be useful if storage space is limited; however, there is a processing cost to compressing and decompressing files. Similarly, it might be quicker to read and write compressed files across networks, but the files subsequently need to be decompressed. There are, then, trade-offs that must be considered before applying this system option.

The amount by which data can be compressed depends on the type of data to be compressed.

BINARY

Use Ross Data Compression (RDC) algorithm.

CHAR

Use run-length encoding (RLE) algorithm.

N

Do not compress.

NO

Do not compress.

ON

Use run-length encoding (RLE) algorithm.

Y

Use run-length encoding (RLE) algorithm.

YES

Use run-length encoding (RLE) algorithm.

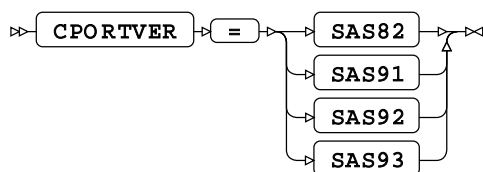
Example

In this example, the `OPTIONS` statement is used to that RDC is used to compress datasets.

```
OPTIONS COMPESS = BINARY;
```

CPORTVER

Specifies the type of `CPORT` file that is generated by default by the `CPORT` procedure.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `SAS92`

Option group: `SASFILES`

Portable: `True`

Restrictable: `True`

Saveable: `True`

SAS82

Generate a file in SAS version 8.2 format.

SAS91

Generate a file in SAS version 9.1 format.

SAS92

Generate a file in SAS version 9.2 format.

SAS93

Generate a file in SAS version 9.3 format.

Example

In this example, the `OPTIONS` statement is used to specify that a version 8.2 transport file is created.

```
OPTIONS CPORTVER = SAS82;
```

DIRECTIO

Specifies whether to use direct input and output, where possible.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NODIRECTIO</code>
Option group:	<code>SASFILES</code>
Portable	True
Restrictable	True
Saveable	True
Supported platform:	AIX for pSeries 64-bit Linux for ARM Linux for pSeries Linux (LE) for pSeries Linux for System z 64-bit Linux for System z 64-bit Linux 32-bit Linux 64-bit Mac O/S Solararis for SPARC Solaris for 64-bit x86 Solaris for 32-bit x86 64-bit Windows 32-bit Windows

DIRECTIO

Use direct input and output.

NODIRECTIO

Do not use direct input and output.

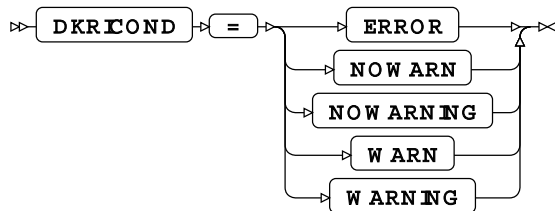
Example

In this example, the `OPTIONS` statement is used to specify that direct I/O is not used.

```
OPTIONS NODIRECTIO
```

DKRCOND

Specifies the action to take for `DROP`, `KEEP`, and `RENAME` error conditions on input datasets.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>ERROR</code>
Option group:	<code>SASFILES</code>
Portable	True
Restrictable	True
Saveable	True

ERROR

The program stops, and an error message is written to the log.

NOWARN

The program continues, and no warning message is written to the log.

NOWARNING

The program continues, and no warning message is written to the log.

WARN

The program continues, but a warning message is written to the log.

WARNING

The program continues, but a warning message is written to the log.

Example

In this example, the `OPTIONS` statement is used to specify that a warning is generated on `DROP`, `KEEP`, and `RENAME` error conditions, but the program continues to run.

```
OPTION DKRCOND=WARN;
```

If you then run the following program, in which the specified input dataset has no variable named `C`, no error occurs, but a warning is written to the log

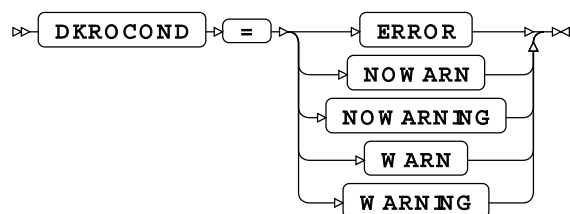
```
LIBNAME test "c:\temp" ;  
DATA testout;  
  SET test.books (drop = C);  
RUN;
```

The log contains the following lines:

```
49      data testout;  
150      set test.books (drop = C);  
WARNING: Variable "C" referenced in the KEEP or DROP list is not known  
151      run;
```

DKROCOND

Specifies the action to take for `DROP`, `KEEP`, and `RENAME` error conditions on output datasets.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `WARN`

Option group: `SASFILES`

Portable: `True`

Restrictable: `True`

Saveable: `True`

ERROR

The program stops, and an error message is written to the log.

NOWARN

The program continues, and no warning message is generated.

NOWARNING

The program continues ,and no warning message is generated.

WARN

The program continues, but a warning message is written to the log.

WARNING

The program continues, but a warning message is written to the log.

Example

In this example, the `OPTIONS` statement is used to specify that a warning is generated on `DROP`, `KEEP`, and `RENAME` error conditions, but the program continues to run.

```
OPTION DKRICOND=WARN;
```

If you then run the following program, in which the specified output dataset has no variable named `C`, no error will occur, but a warning is written to the log

```
LIBNAME test "c:\temp" ;
DATA testout (rename = (C = D));
  SET test.books;
RUN;
```

The log contains the following:

```
WARNING: The variable "C" in the RENAME list has never been referenced
```

DLCREATEDIR

Specifies that a folder is created when a libname is assigned, if the folder does not already exist.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NODLCREATEDIR</code>
Option group:	<code>SASFILES</code>
Portable	False
Restrictable	True
Saveable	False

DLCREATEDIR

Create a folder.

NODLCREATEDIR

Do not create a folder.

Basic example

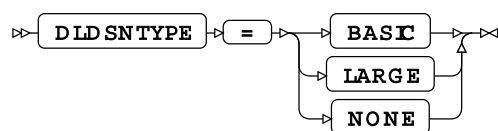
In this example, the `OPTIONS` statement is used to specify that a new folder is created for a libname if the folder does not exist.

```
OPTIONS DLCREATEDIR;  
LIBNAME moretemp 'c:\temp\newtemp';
```

The folder `newtemp` is created in `c:\temp`.

DLDSNTYPE

Specifies the default value to use for the `DSNTYPE` option of a `LIBNAME` connection statement.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NONE
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

BASIC

Specifies a basic format dataset.

LARGE

Specifies a large format dataset.

NONE

The library is allocated without specifying `DSNTYPE`

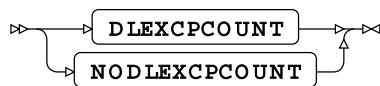
Example

In this example, the `OPTIONS` statement is used to specify a `DSNTYPE` of `LARGE`.

```
OPTIONS DLDSNTYPE = LARGE;
```

DLEXCPCOUNT

Specifies whether to report the EXCP count for WPS data libraries.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NODLEXCPCOUNT</code>
Option group:	<code>SASFILES</code>
Portable	False
Restrictable	True
Saveable	False
Supported platform:	z/OS for System z

DLEXCPCOUNT

Report EXCP count.

NODLEXCPCOUNT

Do not report EXCP count.

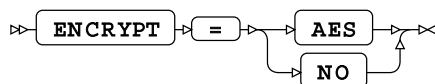
Example

In this example, the `OPTIONS` statement is used to specify that the EXCP count is not reported.

```
OPTIONS NODLEXCPCOUNT;
```

ENCRYPT

Specifies whether datasets are encrypted.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
-----------	--

Default:	NO
Maximum length:	10
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

File are encrypted using the Advanced Encryption Standard (AES256). AES encryption requires a key, which can be specified using the `ENCRYPTKEY` system option.

AES

Encrypt datasets using AES.

NO

Do not encrypt datasets.

To encrypt a file you need to specify both `ENCRYPT` and `ENCRYPTKEY`. To decrypt a file you need only specify `ENCRYPTKEY`. You can then read the contents of the dataset and manipulate the values. If you want to read an encrypted file and write the contents of that file to a dataset unencrypted, you need to specify `ENCRYPTKEY` with the corresponding key, and `ENCRYPT = NO`. `ENCRYPT = NO` is set by default when you start a WPS session.

Note:

This system option provides the same functionality as the `ENCRYPT` dataset option.

Basic example

In this example, the `OPTIONS` statement is used to specify that files are encrypted.

```
OPTIONS ENCRYPT=AES ENCRYPTKEY = 'xxyy99';
LIBNAME books XLSX "c:\temp\books\books.xlsx";
LIBNAME books2 "c:\temp\books";
DATA books2.booksout;
    SET books.books1;
RUN;
```

The dataset `booksout` is encrypted using the key `xxyy99`.

Example – read encrypted dataset, write unencrypted dataset

In this example, the `OPTIONS` statement is used to specify that files are decrypted. This example uses the encrypted file created in the previous example, and assumes the session has not been restarted. To write an unencrypted dataset you therefore need to specify `ENCRYPT = NO`.

```
OPTIONS ENCRYPT=NO;
DATA temp;
    SET books2.booksout;
    IF author = 'Wilson, Colin' THEN OUTPUT;
RUN;
```

The specified dataset `booksout` is decrypted using the key `xyxy99` which is available throughout the session. The working dataset `temp` is unencrypted.

ENCRYPTKEY

Specifies the key used to encrypt and decrypt WPD datasets.

⇒ **ENCRYPTKEY** ⇒ = ⇒ *encryption-key* ⇒ ⇐

Valid in:	OPTIONS statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	32000
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

encryption-key

The encryption key. This can contain up to 32,000 characters.

The characters in *encryption-key* are represented using the current character encoding for the session. You cannot, therefore, decrypt an encrypted dataset in a session that uses a different character encoding if the key contains characters whose representation differs in that encoding. If you need to transfer an encrypted dataset between machines with different session encodings, we recommend you use a hexadecimal-encoded encryption key to ensure the key is the same in all session encodings.

Note:

If the encryption-key is lost you cannot recover encrypted data. You must therefore ensure the encryption key is securely stored. WPS does not record the encryption key and is unable to recover a key that has been lost.

Note:

This system option provides the same functionality as the `ENCRYPTKEY` dataset option.

Basic example

In this example, the `OPTIONS` statement is used to specify that datasets are encrypted.

```
OPTIONS ENCRYPT=AES ENCRYPTKEY = 'xyxy99';
LIBNAME books XLSX "c:\temp\books\books.xlsx";
LIBNAME books2 "c:\temp\books";
DATA books2.booksout;
    SET books.books1;
RUN;
```

The dataset `booksout` is encrypted using the key `xyxy99`.

Example – read encrypted dataset

In this example, the `OPTIONS` statement is used to specify that datasets are decrypted. This example uses the file that was encrypted in the previous example.

```
OPTIONS ENCRYPTKEY = 'xyxy99';
LIBNAME books2 "c:\temp\books";
DATA _NULL_;
    SET books2.booksout;
    IF author = 'Wilson, Colin' THEN PUT title;
RUN;
```

The specified dataset `booksout` is decrypted using the key `xyxy99`.

ENGINE

Specifies the default library engine or data engine to use.

⇒ **ENGINE** ⇒ = ⇒ *engine-name* ⇒

Valid in:	Configuration file and command line.
Default:	WPD
Maximum length:	8
Option group:	SASFILES
Portable	True
Restrictable	False
Saveable	False

engine-name

The name of an engine. This can contain up to eight characters.

The available engines are listed in *Library engines* and *Data Engines*.

Example

In this example, the option is specified on the command line, and sets the library engine as `ACCESS`.

```
wps c:\temp\test.wps -engine access
```


The program `test.wps` is:

```
LIBNAME betest "c:\temp\test1.accdb";
DATA betest.table1;
  INPUT txt;
  FORMAT txt 2.;
CARDS;
6
;
```

This writes the data to the table `table1` in the specified Microsoft Access database.

FILEDEV

Specifies the default device to be used for new physical files on z/OS.

➤ **FILEDEV** ➤ = ➤ ***device-name*** ➤

Valid in: `OPTIONS` statement, configuration file and command line.

Default: `SYSDA`

Maximum length: `8`

Option group: `EXTFILES`
 `SASFILES`

Supported platform: `z/OS for System z`

device-name

The name of the default device. This can be up to eight characters long.

Example

In this example, the `OPTIONS` statement is used to specify that the default device is `VIO`.

```
OPTIONS FILEDEV=VIO;
```

FILEMSGSGS

Specifies whether to write messages to the log about the results of dynamic allocations using `DDNAME`.

➤ **FILEMSGSGS** ➤
➤ **NOFILEMSGSGS** ➤

Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOFILEMSGSGS`

Option group: EXTFILES
 SASFILES

Supported platform: z/OS for System z

FILEMSGSGS

Write messages to the log.

NOFILEMSGSGS

Do not write messages to the log.

A message written to the log during dynamic allocation has the following form:

```
1 message from dynalloc
1GD103I SMS ALLOCATED TO DDNAME CAT SYS0008
```

Example

In this example, the `OPTIONS` statement is used to specify that dynamic allocation log messages are not display.

```
OPTIONS NOFILEMSGSGS;
```

FILESPPRI

Specifies the default primary space allocation for new physical files.

⇒ **FILESPPRI** ⇒ = ⇒ *default-space* ⇐

Valid in: `OPTIONS` statement, configuration file and command line.

Default: 1

Minimum value: 1

Maximum value: 32760

Option group: EXTFILES
 SASFILES

Supported platform: z/OS for System z

default-space

The default primary space allocation.

The space is allocated in the units specified by the system option `FILEUNIT` [↗](#) (page 147).

Example

In these examples, the `OPTIONS` statement is used to specify the size of the primary space allocation.

```
OPTIONS FILEUNIT = CYL FILESPPRI = 2;
```

This sets the default primary space allocation to two cylinders.

```
OPTIONS FILEUNIT = 1024 FILESPPRI = 56;
```

This sets the default primary space allocation to 56 units of 1024 bytes, or 57,344 bytes.

FILESPSEC

Specifies the default secondary space allocation for new physical files.

➤ **FILESPSEC** = *default-space* ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	1
Minimum value:	0
Maximum value:	32760
Option group:	EXTFILES SASFILES
Supported platform:	z/OS for System z

default-space

The default secondary space allocation.

The space is allocated in the units specified by the system option `FILEUNIT` [↗](#) (page 147).

Example

In these examples, the `OPTIONS` statement is used to specify the size of the secondary space allocation.

```
OPTIONS FILEUNIT = CYL FILESPPRI = 1;
```

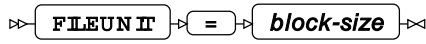
This sets the default secondary space allocation to one cylinders.

```
OPTIONS FILEUNIT = 1024 FILESPPRI = 56;
```

This sets the default secondary space allocation to 57,344 bytes.

FILEUNIT

Specifies the default unit of allocation for new physical files.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	CYL
Maximum length:	8
Option group:	EXTFILES SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

block-size

The default block-size. The size can be specified as a number of bytes, or as a string. The string can be:

BLKS or BLK	Blocks
CYLS or CYL	Cylinders
TRKS or TRK	Tracks

This option enables you to specify the default unit of allocation as either a number of bytes or as an element of disk storage, such as a cylinder or track. These units can then be specified to other system options, such as FILESPPRI and FILESPSEX. For example, you might set the default unit of allocation as blocks. You can then use this as the unit to when specifying FILESPPRI. Setting FILESPPRI = 16 would then set the primary space allocation for new physical files to 16 blocks. The number of bytes that this specifies depends on the type of device to which the allocation applies. If you specify the default unit as a number of bytes, that number of bytes is used as the unit. For example, you might set the default unit of allocation as 1024 bytes; specifying FILESPPRI = 24 would then set the primary space allocation for new physical files to 24576 bytes.

Example

In this example, the OPTIONS statement is used to specify the default unit of allocation.

```
OPTIONS FILEUNIT = TRK;
```

The default unit of allocation is tracks.

```
OPTIONS FILEUNIT = 2048;
```

The default unit of allocation is 2048 bytes.

FIRSTOBS

Sets the number of the first observation to process in a dataset.

➤ **FIRSTOBS** ➤ = ➤ *observation-number* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	1
Minimum value:	1
Maximum value:	Maximum integer value supported by the system.
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

observation-number

The number of the first observation to process. This can be specified as:

- The number of the observation; for example, you can enter 5.
- The number of the observation as a multiple of 1,024, $1,024^2$ or $1,024^3$, by appending K, M or G to the value, respectively. For example, if the value is 0.5K, the first observation read is the 512th in the dataset.
- The number of the observation specified in hexadecimal, by appending x. For example, if the value is BBx, the first observation is 187th in the dataset.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

You can use OBS and FIRSTOBS to specify a range of observations to read from a dataset. For example, if you set `OPTIONS FIRSTOBS=10 OBS=30`, twenty observations are read, starting at the tenth observation of the dataset and ending at the 30th.

Example

In this example, the `OPTIONS` statement is used to specify that the dataset is read from the fifth observation onward.

```
OPTIONS FIRSTOBS = 5;
```

IBUFNO

Specifies the number of index file buffers used by a library engine for a dataset.

➤ **IBUFNO** ➤ = ➤ *buffer-number* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0
Minimum value:	0
Maximum value:	10000
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

buffer-number

The total amount of file index buffers available. This can be specified as:

- The number of buffers; for example, you can enter 5000.
- The number of buffers as a multiple of 1024, by appending K to the value. For example, if the value is 7K, 7,168 index file buffers are used. This number must be an integer.
- The of buffers specified in hexadecimal, by appending X. For example, if the value is 1000X, the number of buffers is 4096.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

Example

In this example, the OPTIONS statement is used to specify that 5000 index file buffers are available.

```
OPTIONS IBUFNO = 5000;
```

IBUFSIZE

Specifies the size of an index page for a WPS dataset.

➤ **IBUFSIZE** ➤ = ➤ *buffer-size* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	0

Minimum value:	0
Maximum value:	32767
Option group:	SASFILES PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

buffer-size

The size of index page to use, in bytes. This can be specified as:

- The number of bytes; for example, you can enter 5000.
- The number of kibibytes, by appending K to the value. For example, if the value is 7K, the index page size is 7,168 bytes. The number must be an integer.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

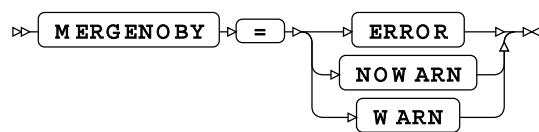
Example

In this example, the `OPTIONS` statement is used to specify that the index page is eight kibibytes in size.

```
OPTIONS IBUFNO = 8K;
```

MERGENOBY

Specifies whether to generate an error, issue a warning, or not issue a warning when no `BY` statement is provided with a `MERGE` statement.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOWARN
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

ERROR

Generate an error. The program stops, and a message is written to the log.

NOWARN

No warning is issued, and no error generated.

WARN

Write a warning to the log.

If you set `WARN` or `ERROR`, and no `BY` statement is specified for the `MERGE`, the following message is written to the log:

```
The MERGE statement at line nnn column cc has no corresponding BY statement.
```

nnn is the line number and *cc* is the column number of the `MERGE` statement. The message is prefixed with `WARNING` or `ERROR` appropriately.

Example

In this example, the `OPTIONS` statement is used to specify that the program stops and a message is written to the log if no `BY` statement is specified.

```
OPTIONS MERGENOBY=ERROR;
LIBNAME books "c:\temp\books";
DATA test;
    MERGE books.books books.sortedbks;
RUN;
```

The following error message is written to the log:

```
ERROR: The MERGE statement at line 220 column 5 has no corresponding BY statement.
NOTE: DATA step was not executed because of errors detected
```

OBS

Specifies the number of the last observation to process in a dataset.

```
>> [OBS] [=] [observation-number] <<
```

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	9.223372036854776e+18
Minimum value:	0
Maximum value:	Maximum integer value supported by the system.
Option group:	<code>SASFILES</code>
Portable	True
Restrictable	True
Saveable	True

observation-number

The number of the observation. This can be specified as:

- The number of the observation; for example, you can enter 2147488.
- The number of the observation as a multiple by 1024, by 1024^2 or 1024^3 , by appending K, M or G to the value, respectively. For example, if the value is 1M, the observation read is the 1,048,576th in the dataset. Decimal values are rounded.
- A number of the observation specified in hexadecimal, by appending x. For example, if the value is BEBC2X, the number of lines read is 781250.
- MIN – the minimum supported value.
- MAX – the maximum supported value.

You can use OBS and FIRSTOBS to specify a range of observations to read from a dataset. For example, if you set `OPTIONS FIRSTOBS=11 OBS=30`, twenty observations are read, starting at the 11th observation of the dataset and ending at the 30th.

If you set OBS to 0 (zero) or MIN, any subsequent DATA step that reads a dataset does not execute any instructions in that step. Any output dataset is created with appropriate variables, but no data is written to it.

Example

In this example, the `OPTIONS` statement that the dataset is read from the first observation until the tenth.

```
OPTIONS OBS = 10;
```

REPLACE

Specifies whether a permanent dataset can be replaced.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOREPLACE
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

REPLACE

Allow dataset replacement.

NOREPLACE

Do not allow dataset replacement.

A permanent dataset is a dataset created anywhere except the WPS *Work* library.

If **NOREPLACE** is set and the dataset to which you are writing already exists, a warning message is written to the log. This has the format:

```
WARNING: The dataset "dsname" was not replaced because of the NOREPLACE option
```

dsname is the name of the dataset.

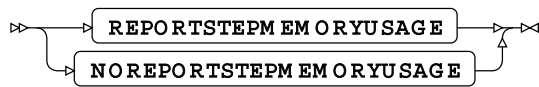
Example

In this example, the **OPTIONS** statement is used to specify that datasets cannot be replaced.

```
OPTIONS NOREPLACE;
```

REPORTSTEPMEMORYUSAGE

Specifies whether to report memory usage for a program step.



Valid in: **OPTIONS** statement, configuration file and command line.

Default: **NOREPORTSTEPMEMORYUSAGE**

Option group: **SASFILES**

Portable: **True**

Restrictable: **True**

Saveable: **True**

Supported platform: **z/OS for System z**

REPORTSTEPMEMORYUSAGE

Report memory usage after each step.

NOREPORTSTEPMEMORYUSAGE

Do not report memory usage after each step.

If this option is set, the memory usage for a preceding program step and the amount of time used is reported.

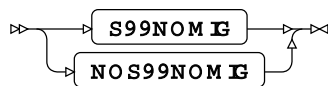
Example

In this example, the `OPTIONS` statement is used to specify that memory use is not reported.

```
OPTIONS NOREPORTSTEPMEMORYUSAGE;
```

S99NOMIG

Specifies whether to restore migrated datasets.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOS99NOMIG`

Option group: `EXTFILES`
`SASFILES`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

S99NOMIG

Restore migrated data sets.

NOS99NOMIG

Do not restore migrated data sets.

Example

In this example, the `OPTIONS` statement is used to specify that migrated datasets are not restored.

```
OPTIONS NOS99NOMIG;
```

SEQENGINE

Specifies the default library engine for sequential dataset files.



Valid in: `OPTIONS` statement, configuration file and command line.

Default:	WPDSEQ
Maximum length:	8
Option group:	SASFILES
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

library-engine

The name of the library engine.

The library engine names are described in *Library engines*.

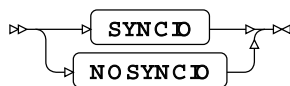
Example

In this example, the `OPTIONS` statement is used to specify that the SASSEQ data engine is used.

```
OPTIONS SEQENGINE=SASSEQ;
```

SYNCIO

Specifies whether to use synchronous input and output to access datasets.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	NOSYNCIO
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

SYNCIO

Use operating system synchronous input or output.

NOSYNCIO

Do not use operating system synchronous input or output.

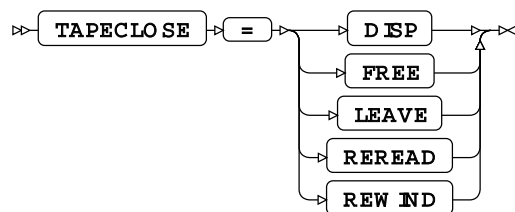
Example

In this example, the `OPTIONS` statement is used to specify operating system input or output is not used.

```
OPTIONS NOSYNCRIO;
```

TAPECLOSE

Specifies the default behaviour of sequential access boundaries for tape libraries when a library dataset is closed.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `REREAD`

Option group: `SASFILES`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

DISP

Dispose of a tape volume as specified by the DD statement associated with the dataset.

FREE

Free the current dataset when it is closed, rather than at the end of the job step. The volume can be used by other tasks or can be demounted.

LEAVE

Position the current volume at the logical end of the dataset. If processing was forward, this is the end of the portion of the dataset residing on the current volume. If processing was backward this is at the beginning of the portion of the dataset residing on the current volume. If `FREE=CLOSE` is specified in the JCL, the dataset is not unallocated until the end of the job step.

REREAD

Position the current volume for reprocessing. If processing was forward, the volume is positioned at the beginning of the dataset. If processing was backward the volume is positioned at the end of the dataset. If `FREE=CLOSE` is specified in the JCL, the dataset is not unallocated until the end of the job step.

REWIND

Position the volume at the load point, whatever the direction of processing.

For more information on these options, see your IBM z/OS documentation.

Example

In this example, the `OPTIONS` statement is used to specify that tape volumes are freed when the dataset is closed.

```
OPTIONS TAPECLOSE=FREE;
```

TAPEENGINE

Specifies the library engine to use when the `TAPE` option is specified in `LIBNAME`.

» **TAPEENGINE** = *library-engine* «

Valid in:	Configuration file and command line.
Default:	WPSTAPE
Maximum length:	8
Option group:	SASFILES
Portable	True
Restrictable	False
Saveable	False

library-engine

The engine name to use.

`TAPE` is a pseudo-engine name that can be used in a `LIBNAME` statement. If `TAPE` is specified, `TAPEENGINE` must also be set.

Example

In this example, the option is specified on the command line.

```
wps c:\temp\test.wps -tapeengine access
```

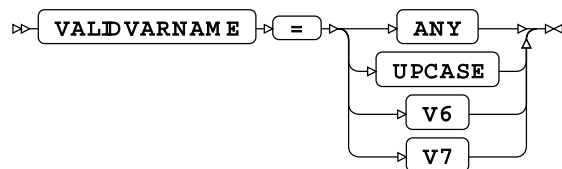
The program `test.wps` is:

```
LIBNAME te TAPE "c:\temp";
DATA te.test_out;
  INPUT txt $10.;
CARDS;
Helloworld
;
```

This writes the data to the dataset `test_out.sas7bdat` in the specified folder.

VALIDVARNAME

Specifies how to treat variable names in data sources that have invalid names.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	V7
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

Data sources such as datasets, spreadsheets and database tables might contain variable names that were created using characters not allowed in SAS-language variable names. For example, a dataset might contain variables that contain spaces as separators: `Book Title or Number of Cases`. By specifying this system option, you specify how input variables are treated in programs.

ANY

Any text can be a valid variable name.

If you specify this option, any characters, including reserved characters, can be specified in variable names. The variable names are written as specified to a dataset. However, if you want to work with the variable in a program, you need to specify the variable as a string literal.

For example, if you specify this option, and an input dataset variable is named `Type/of/Book`, the variable is written to an output dataset with this name. However, if you want to set the variable in the `DATA` step, you have to specify it as a string literal; for example:

```
IF 'Type/of/Book'n = 'SF' THEN OUTPUT;
```

UPCASE

A variable name containing invalid characters is replaced with a valid V7 variable name. Lower case alphabetic characters are replaced with upper case.

To create a valid V7 variable name, any invalid characters are replaced with an underscore. A valid V7 variable name can be longer than eight characters, so the variable name is not truncated.

For example, if you specify this option, and an input dataset variable is named `Type of Book`, the variable is written to an output dataset with the name `TYPE_OF_BOOK`. Spaces are invalid in a V7 variable name. If you want to set the variable in the `DATA` step, you can specify it in either upper case or lower case, as the SAS language ignores the case of variables, and include the underscores. For example:

```
IF type_of_book = 'SF' THEN OUTPUT;
```

V6

A variable name containing invalid characters is replaced with a valid V6 variable name. To create a valid V6 variable name, any invalid characters are replaced with an underscore. A valid V6 variable name cannot be longer than eight characters; if the variable name is longer than this, it is truncated. If the final character of the truncated name is an invalid character, it is removed. A variable name longer than eight characters might, therefore be truncated so that it is shorter than eight characters. If the final character of the truncated name is an invalid character, but is the last of a series of invalid characters, the name is truncated to eight characters.

For example, if you specify this option, and an input dataset variable is named `Type of Book`, the variable is written to an output dataset with the name `Type_of`. Spaces are invalid in a V6 variable name. If you want to set the variable in the `DATA` step, you must include the underscores. For example:

```
IF type_of = 'SF' THEN OUTPUT;
```

V7

For both V6 and V7, if the first character of a variable name is a numeral, an underscore is appended to the name. For example, if the variable name is `1Type of Book`, the V6 variable name is `_1Type_o`, and the V7 variable name is `_1Type_of_Book`.

Example

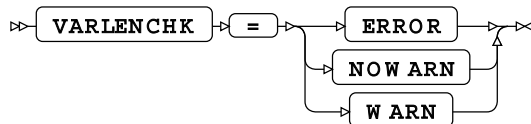
In this example, the `OPTIONS` statement is used to specify that any text string can be used as a variable name.

```
LIBNAME NEWBOOKS XLSX "c:\temp\books\books_new.xlsx";
OPTIONS VALIDVARNAME='any';
DATA temp;
  SET newbooks.books;
  if '1type of book'n = 'SF' then output;
run;
```


In this example, the variable `1Type` of `Book` in the Excel worksheet is written to the dataset `temp`. In the `DATA` step the variable is specified as a literal string.

VARLENCHK

Controls the behaviour in the `DATA` step when variables from different input datasets have different lengths.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOWARN
Option group:	SASFILES
Portable	True
Restrictable	True
Saveable	True

This system option specifies what happens if datasets that are being merged or concatenated contain the same variable, but that variable has different lengths in the datasets. Whether an error occurs is firstly dependent on the order in which the datasets are specified, secondly on this system option. For example, if the variable is longer in the first dataset defined than it is in the second dataset defined, no error occurs. However, if the variable is shorter in the first dataset defined than it is in the second dataset defined, you can specify that an error or warning occurs using this system option.

ERROR

An error occurs. A message is written to the log and the program stops.

NOWARN

No messages are written to the log; the program continues running.

WARN

A warning message is written to the log but the program continues running.

For both `WARN` and `ERROR`, a message is written to the log:

```
Multiple lengths were specified for the variable author by input data set(s). This  
may cause truncation
```

Example

In this example, the `OPTIONS` statement is used to specify that a warning is generated if the variable in the first dataset is shorter than the corresponding variable in the second.

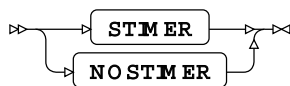
```
OPTIONS VARLENCHK=WARN;  
DATA OUT;  
    SET temp2 temp;  
RUN;
```

SMF group system options

SMF related settings

STIMER

Specifies whether to write performance statistics to the log after each step.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSTIMER</code>
Option group:	<code>PERFORMANCE</code> <code>SMF</code>
Portable	False
Restrictable	True
Saveable	False

STIMER

Write performance statistics after each step.

NOSTIMER

Do not write performance statistics after each step.

A step can be the execution of a procedure or `DATA` step.

Example

In this example, the `OPTIONS` statement is used to specify that performance statistics are not written to the log.

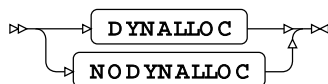
```
OPTIONS NOSTIMER;
```

SORT group system options

System options that specify options the control the `SORT` procedure and external programs.

DYNALLOC

Specifies whether a host utility is assumed to support dynamic allocation of work files.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NODYNALLOC`

Option group: `SORT`

Portable: `True`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

DYNALLOC

Assume the host utility supports work file dynamic allocation.

NODYNALLOC

Assume the host utility does not support work file dynamic allocation.

If you specify `DYNALLOC` and the host utility doesn't support it, an error occurs.

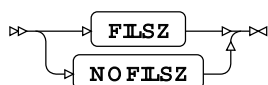
Example

In this example, the `OPTIONS` statement is used to specify the host utility does not support work file dynamic allocation.

```
OPTIONS NODYNALLOC;
```

FILSZ

Specifies whether to use `FILSZ` in the host sort control string.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOFILSZ
Option group:	SORT
Portable	False
Restrictable	True
Saveable	False
Supported platform:	z/OS for System z

FILSZ

Use FILSZ.

NOFILSZ

Do not use FILSZ.

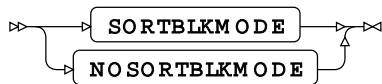
Example

In this example, the OPTIONS statement is used to specify that FILSZ is not used.

```
OPTIONS NOFILSZ;
```

SORTBLKMODE

Specifies whether the sort program supports a block mode interface.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSORTBLKMODE
Option group:	SORT
Portable	False
Restrictable	True
Saveable	True

SORTBLKMODE

The sort program supports a block mode interface.

NOSORTBLKMODE

The sort program does not support a block mode interface.

If you specify this option but the sort program does not support a block mode interface, an error occurs.

Example

In this example, the `OPTIONS` statement is used to specify that the sort program does not support a block mode interface.

```
OPTIONS NOSORTBLKMODE;
```

SORTCHECK

Specifies whether the sort program checks that its output is correctly ordered.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSORTCHECK</code>
Option group:	<code>SORT</code>
Portable	True
Restrictable	True
Saveable	True

SORTCHECK

Check the output.

NOSORTCHECK

Do not check the output.

If the output is checked and it is not ordered correctly, an error occurs.

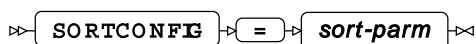
Example

In this example, the `OPTIONS` statement is used to specify that the sort order is not checked.

```
OPTIONS NOSORTCHECK;
```

SORTCONFIG

Specifies the configuration parameters used by the external sort program.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
-----------	--

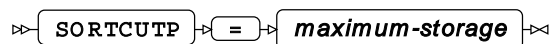
Maximum length:	1024
Option group:	<code>SORT</code>
Portable	True
Restrictable	True
Saveable	True

sort-parm

The configuration parameters used by the external sort program. The parameters are names, or a combination of names and values. The parameters are separated by spaces.

SORTCUTP

Specifies the amount of storage above which the host sort utility is used.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	4194304
Minimum value:	0
Maximum value:	2147483647
Option group:	<code>SORT</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

maximum-storage

The number of bytes to use. This can be specified as:

- The number of bytes; for example, you can enter 2147488.
- The number of kibibytes, mebibytes or gibibytes, by appending `K`, `M` or `G` to the value, respectively. For example, if the value is 100M, the number of bytes is 100MiB. The number specified must be an integer.
- The number of bytes specified in hexadecimal, by appending `x`. For example, if the value is BEBC2X, the number of bytes is 781250.
- `MIN` – the minimum supported value.
- `MAX` – the maximum supported value.

The default sort utility used by the SORT procedure is used until the value specified by *maximum-storage* is exceeded. The host sort utility will then be used instead.

Example

In this example, the `OPTIONS` statement is used to specify that the host sort utility is used once more than 1GiB of memory is used.

```
OPTIONS SORTCUTP = 1G;
```

SORTDEV

Specifies the device on which WPS allocates sort work files before calling the host sort.

➤ `SORTDEV` ➤ = ➤ `device-name` ➤

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<empty-string>
Maximum length:	8
Option group:	<code>SORT</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

When the sort utility of the host is used, and WPS is configured to allocate work files, the device specified by this option is used for work files.

device-name

The name of the device to use.

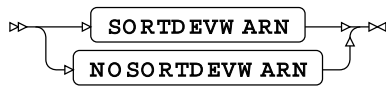
Example

In this example, the `OPTIONS` statement is used to specify the that the SYSDA device is used for work files.

```
OPTIONS SORTDEV = SYSDA;
```

SORTDEVWARN

Specifies whether to write a warning to the log if the `SORTDEV` system option contains the name of a device group rather than a specific device.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSORTDEVWARN
Option group:	SORT
Portable	True
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

SORTDEVWARN

Write warning.

NOSORTDEVWARN

Do not write warning.

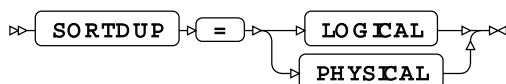
Example

In this example, the `OPTIONS` statement is used to specify that a warning is not written.

```
OPTIONS NOSORTDEVWARN;
```

SORTDUP

Specifies whether the `NODUP` option to the `SORT` is applied to physical or logical records



Valid in:	OPTIONS statement, configuration file and command line.
Default:	PHYSICAL
Option group:	SORT
Portable	True
Restrictable	True
Saveable	True

LOGICAL

Remove only duplicates that remain after any `DROP` and/or `KEEP` dataset options are processed.

PHYSICAL

Remove duplicates based on all the variables present.

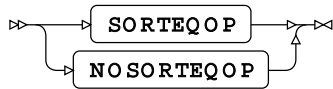
Example

In this example, the `OPTIONS` statement is used to specify that logical duplicates are removed.

```
OPTIONS SORTDUP = PHYSICAL;
```

SORTEQOP

Specifies whether the host sort routine implements the `EQUAL` option.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOSORTEQOP`

Option group: `SORT`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

Enabling the source routine to implement the `EQUALS` option can improve performance.

SORTEQOP

The host sort routine implements the option.

NOSORTEQOP

The host sort routine does not implement the option.

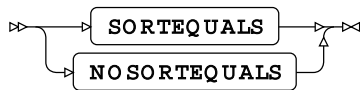
Example

In this example, the `OPTIONS` statement is used to specify that the host sort routine implements the option.

```
OPTIONS SORTEQOP;
```

SORTEQUALS

Specifies whether the order of observations with the same BY value is maintained in the `Sort` procedure.



Valid in: `Options` statement, configuration file and command line.

Default: `NOSORTEQUALS`

Option group: `Sort`

Portable: `True`

Restrictable: `True`

Saveable: `True`

SORTEQUALS

The order of observations is maintained.

NOSORTEQUALS

The order of observations is not maintained.

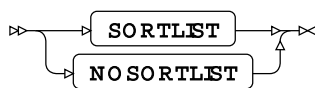
Example

In this example, the `Options` statement is used to specify that the order of observations is not maintained.

```
Options NOSORTEQUALS;
```

SORTLIST

Specifies whether the `List` option is to be specified to the host sort utility.



Valid in: `Options` statement, configuration file and command line.

Default: `NOSORTLIST`

Option group: `Sort`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: z/OS for System z

SORTLIST

Specify the LIST option.

NOSORTLIST

Do not specify the LIST option.

Specifying this system option to the host sort utility determines whether the list of directives defined for the host sort utility is performed.

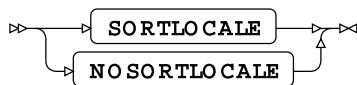
Example

In this example, the `OPTIONS` statement is used to specify that the LIST option is not sent the host sort utility.

```
OPTIONS NOSORTLIST;
```

SORTLOCALE

Specifies whether the host sort routine implements the LOCALE option.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: NOSORTLOCALE

Option group: SORT

Portable False

Restrictable True

Saveable True

Supported platform: z/OS for System z

SORTLOCALE

Implement the option.

NOSORTLOCALE

Do not implement the option

Example

In this example, the `OPTIONS` statement is used to specify that the option is not implemented.

```
OPTIONS NOSORTLOCALE;
```

SORTMAXKEY

Specifies the maximum key length for the host sort routine.

➤ **SORTMAXKEY** ➤ = ➤ *maximum-key-length* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	4084
Minimum value:	1
Maximum value:	32767
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

Total key field length must be less than this value, otherwise the WPS sort is used.

maximum-key-length

The maximum key length to use.

Example

In this example, the `OPTIONS` statement is used to specify the maximum key length for the host sort routine,

```
OPTION SORTMAXKEY=1024;
```

SORTMAXOFF

Specifies the maximum key offset permitted for the host sort routine.

➤ **SORTMAXOFF** ➤ = ➤ *maximum-key-offset* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Default:	4092
Minimum value:	1
Maximum value:	32767
Option group:	SORT
Portable	False
Restrictable	True

Saveable True
Supported platform: z/OS for System z

maximum-key-offset

The maximum key offset permitted for the host sort routine, in bytes.

Example

In this example, the `OPTIONS` statement is used to specify that the maximum key offset permitted for the host sort routine is 8184 bytes.

```
OPTIONS SORTMAXOFF = 8184;
```

SORTMMAP

This system option is provided for compatibility only, and has no effect in WPS.



Valid in: `OPTIONS` statement, configuration file and command line.
Default: NOSORTMMAP
Option group: SORT
Portable True
Restrictable True
Saveable True

SORTMMAP

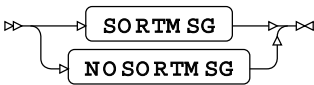
For compatibility only.

NOSORTMMAP

For compatibility only.

SORTMSG

Specifies the `MSG` option value passed to the host sort utility.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSORTMSG
Option group:	SORT
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

SORTMSG

Pass MSG=AP.

NOSORTMSG

Pass MSG=CP.

Example

In this example, the `OPTIONS` statement is used to specify that `MSG=CP` is passed to the host sort utility.

```
OPTIONS NOSORTMSG;
```

SORTNAME

Specifies the name of the host sort utility.

```
<> SORTNAME = <> host-sort-name <>
```

Valid in:	OPTIONS statement, configuration file and command line.
Default:	SORT
Maximum length:	8
Option group:	SORT
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

host-sort-name

The name of the host sort utility.

Example

In this example, the `OPTIONS` statement is used to specify that the name of the host sort routine.

```
OPTIONS SORTNAME = DFSORT;
```

SORTOPTS

Specifies whether an `OPTIONS` statement is generated for the host sort utility.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NOSORTOPTS`

Option group: `SORT`

Portable: `False`

Restrictable: `True`

Saveable: `True`

Supported platform: `z/OS for System z`

This system option specifies that parameters defined to the `SORTPARM` [\(page 361\)](#) system option are set in an `OPTIONS` statement for the host sort utility.

SORTOPTS

Generate an `OPTIONS` statement.

NOSORTOPTS

Do not generate an `OPTIONS` statement.

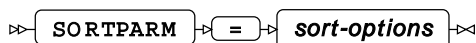
Example

In this example, the `OPTIONS` statement is used to specify that an `OPTIONS` statement is not generated for the host sort utility.

```
OPTIONS NOSORTOPTS;
```

SORTPARM

Specifies additional options to be passed to the host sort utility.



Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	1024
Option group:	SORT
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

sort-options

Parameters to pass to the host sort utility. You specify the parameters as a string that is passed unchanged to the utility.

The parameters are only passed to the utility if you specify SORTOPTS.

Example

In this example, the OPTIONS statement is used to specify sort utility options.

```
OPTIONS SORTPARM = "DYNALLOC, ZDPRINT";
```

SORTPGM

Specifies the sort program to be used by the SORT procedure.

```
>> SORTPGM = sort-name <<
```

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	8
Option group:	SORT
Portable	False
Restrictable	True
Saveable	True

Although this system option can be used on any operating system, it currently only has an effect on z/OS for System z.

sort-name

A string that specifies the sort program, and is one of:

- *sort-name* – Specifies the name of an alternative sort program.
- BEST – Specifies that the host or internal sort program is used, depending on the setting of the SORTCUTP [↗](#) (page 352) system option.

- **HOST** – Specifies that the host sort program is used.
- **SAS** – Specifies that the internal sort program is used.
- **WPS** – Specifies that the internal sort program is used.

The default value for z/OS operating systems is **BEST**; for all other operating systems the default value is **WPS**.

Example

In this example, the **OPTIONS** statement is used to specify the sort program.

```
OPTIONS SORTPGM = HOST;
```

SORTSEQ

Specifies the default collation sequence for the **SORT** procedure.

➤ **SORTSEQ** ➤ = ➤ *collation-sequence* ➤

Valid in:	OPTIONS statement, configuration file and command line.
Maximum length:	32
Option group:	SORT
Portable	True
Restrictable	True
Saveable	True

collation-sequence

A string that specifies a sequence. This must be one of the strings specified in the **SORTSEQ** option of the **SORT** procedure.

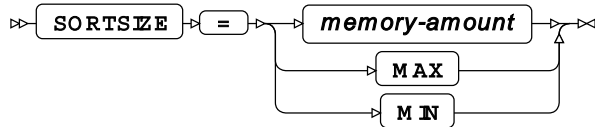
Example

In this example, the **OPTIONS** statement is used to specify that the EBCDIC collation sequence is used for sorting.

```
OPTIONS SORTSEQ = EBCDIC;
```

SORTSIZE

Specifies the amount of memory to use when performing a sort.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	MAX
Maximum length:	32
Option group:	SORT MEMORY PERFORMANCE
Portable	True
Restrictable	True
Saveable	True

memory-amount

The total amount of memory used for the sort, in bytes. This can be specified as:

- The number of bytes; for example, you can enter 2147483648.
- The number of kibibytes, mebibytes or gibibytes, by appending K, M or G to the value, respectively. For example, if the value is 2048M, the memory available for sorting is 2GiB.
- The number, specified in hexadecimal, by appending x. For example, if the value is BEBC200X, the memory available for sorting is 200MB bytes.

The largest number you can enter is 32 characters long.

MAX

The maximum supported value.

MIN

The minimum supported value.

Example

In this example, the `OPTIONS` statement is used to specify that 100MiB of memory is available for a sort.

```
OPTIONS SORTSIZE = 100M;
```

SORTSTATS

Specifies whether to write to the log statistics from the `SORT` procedure steps.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSORTSTATS
Option group:	SORT
Portable	True
Restrictable	True
Saveable	True

SORTSTATS

Write statistics to log.

NOSORTSTATS

Do not write statistics to log.

The following statistics are written to the log:

```

NOTE: Block size   : bsize
Run formation:
  Runs created                : n
  Time reading input dataset  : hh:mm:ss.xxxxx
  Time sorting batches        : hh:mm:ss
  Time outputting batches     : hh:mm:ss
  Total duration              : hh:mm:ss.003989
  Intermediate IO stats:
    Total read volume          : trv B
    Total write volume         : trw B

Merge/output:
  Total duration              : hh:mm:ss.000996
  Intermediate IO stats:
    Total read volume          : trv B
    Total write volume         : trv B

Intermediate file space used : is
  
```

Where:

<i>bsize</i>	The block size used during the sort
<i>n</i>	The number of runs required to sort the data
<i>hh:mm:ss</i>	The time in hours, minutes and seconds
<i>xxxxx</i>	The number of milliseconds
<i>trv</i>	The total amount of data read in bytes

<i>trv</i>	The total amount of data written in bytes
<i>is</i>	The amount of space used for intermediate files

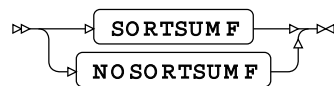
Example

In this example, the `OPTIONS` statement is used to specify that `PROC SORT` statistics are written to the log.

```
OPTIONS SORTSTATS;
```

SORTSUMF

Specifies whether the host sort utility supports the `SUM FIELDS=NONE` option.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSORTSUMF</code>
Option group:	<code>SORT</code>
Portable	False
Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

SORTSUMF

WPS supports the `SUM FIELDS=NONE` option.

NOSORTSUMF

WPS does not support the `SUM FIELDS=NONE`.

Use this option to specify whether the sort utility provided by the operating system supports the `SUM FIELDS=NONE` option. By specifying this option, you define how WPS uses the host sort utility.

If you specify `SORTSUMF` but the host sort utility does not support `SUM FIELDS=NONE`, an error occurs.

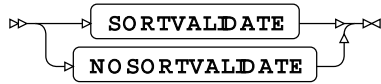
Example

In this example, the `OPTIONS` statement is used to specify that the `SUM FIELDS=NONE` option is not supported by the host sort utility.

```
OPTIONS NOSORTSUMF;
```

SORTVALIDATE

Specifies whether to validate the sort order on datasets with user-defined sort options.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSORTVALIDATE
Option group:	SORT
Portable	True
Restrictable	True
Saveable	True

SORTVALIDATE

Validate sort order.

NOSORTVALIDATE

Do not validate sort order.

You might want to use this option to ensure a sort is correct; for example, to check that the user-defined sort options were sensible, or if a sort has been performed externally using a different collation order.

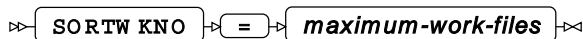
Example

In this example, the `OPTIONS` statement is used to specify that the sort order is not validated.

```
OPTIONS NOSORTVALIDATE;
```

SORTWKNO

Specifies the maximum number of work files to be allocated for sort.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	3
Minimum value:	0
Maximum value:	99
Option group:	SORT
Portable	True

Restrictable	True
Saveable	True
Supported platform:	z/OS for System z

maximum-work-files

The number of work files to be allocated.

Work files are used to store data if the sort process requires more memory than is available.

Example

In this example, the `OPTIONS` statement is used to specify that five workfiles are allocated for sorting.

```
OPTIONS SORTWKNO = 5;
```

SORTWORK

Specifies the location to put SORT procedure work files.

» `SORTW ORK` » `=` » `filepath` »

Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Maximum length:	1024
Option group:	<code>SORT</code>
Portable	True
Restrictable	True
Saveable	True

filepath

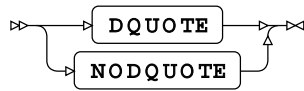
The pathname of the location used to store the work files.

SQL group system options

Specify default values for some `SQL` options.

DQUOTE

This system option is provided for compatibility only, and has no effect in WPS.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NODQUOTE
Option group:	SQL
Portable	True
Restrictable	True
Saveable	True

DQUOTE

NODQUOTE

If you specify `NODQUOTE` a warning message is written to the log.

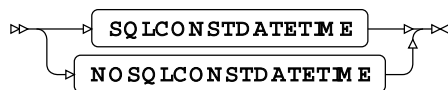
Example

In this example, the `OPTIONS` statement is used to specify that `DQUOTE` is not supported.

```
OPTIONS NODQUOTE;
```

SQLCONSTDATETIME

Specifies whether the SQL `DATE()`, `TIME()` and `DATETIME()` functions are evaluated only once in an SQL query. This option can be restricted.



Valid in:	OPTIONS statement, configuration file and command line.
Default:	NOSQLCONSTDATETIME
Option group:	SQL
Portable	True
Restrictable	True
Saveable	True

SQLCONSTDATETIME

Evaluate functions only once.

NOSQLCONSTDATETIME

Evaluate functions every time.

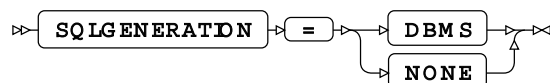
Example

In this example, the `OPTIONS` statement is used to specify that functions are evaluated every time.

```
OPTIONS NOSQLCONSTDATETIME;
```

SQLGENERATION

Specifies whether summary statistics produced by the MEANS and SUMMARY procedures are created by an SQL query on the DBMS, or by WPS.



Valid in: `OPTIONS` statement, configuration file and command line.

Default: `NONE`

Option group: `SQL`

Portable: `True`

Restrictable: `True`

Saveable: `True`

DBMS

Summary statistics are produced by the DBMS using SQL statements.

NONE

Summary statistics are produced by WPS.

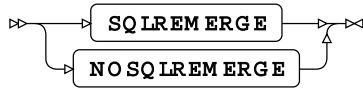
Example

In this example, the `OPTIONS` statement is used to hand off summarisation activity to a DBMS via an SQL query.

```
OPTIONS SQLGENERATION = DBMS;
```


SQLREMERGE

Specifies whether aggregated data can be remerged with the original dataset within the `SQL` procedure.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	<code>NOSQLREMERGE</code>
Option group:	<code>SQL</code>
Portable	True
Restrictable	True
Saveable	True

SQLREMERGE

Remerge aggregated data.

NOSQLREMERGE

Do not remerge aggregated data.

The SQL query planner identifies data to be remerged; if the remerge cannot be performed, the operation stops with an appropriate message.

Example

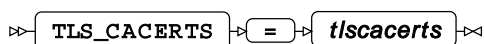
In this example, the `OPTIONS` statement is used to specify that aggregated data cannot be remerged with the original dataset.

```
OPTIONS NOSQLREMERGE;
```

TLS group system options

TLS_CACERTS

Specifies the location of the SSL Trusted Authorities certificate.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
-----------	--

Default:	<empty-string>
Maximum length:	256
Option group:	TLS
Appendable	False
Portable	True
Restrictable	True
Saveable	True
Supported platform:	AIX for pSeries 64-bit Linux for ARM Linux for pSeries Linux (LE) for pSeries Linux for System z 64-bit Linux for System z 64-bit Linux 32-bit Linux 64-bit Mac O/S Solararis for SPARC Solaris for 64-bit x86 Solaris for 32-bit x86 64-bit Windows 32-bit Windows

The Trust Authorities certificates can be used with WPS Hub and SMTP.

tlscacerts

The path to the location.

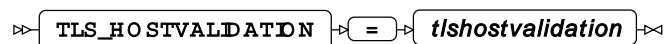
Example

In this example, the `OPTIONS` statement is used to specify the path to the library.

```
OPTIONS TLS_CACERTS = "/etc/ssl/certs/cacert";
```

TLS_HOSTVALIDATION

Specifies whether SSL host key validation is performed.



Valid in:	<code>OPTIONS</code> statement, configuration file and command line.
Default:	SYSTEM
Option group:	TLS

Portable True

SSL host key validation method can be used with SMTP and WPS Hub.

tlshostvalidation

The validation method:

- NONE
- SYSTEM

Example

In this example, the `OPTIONS` statement is used to specify the system method for host key validation.

```
OPTIONS TLS_CACERTS = SYSTEM;
```

Formats

Formats define how output from a SAS language program is displayed.

Formats help WPS understand how to write data values for display. For example, the SAS language represents dates internally as the number of days since 01-January-1960. To display a date as "17-Aug-2015" (or a variation thereof) is much more helpful to a user than to present it as "20317". Formats handle this conversion, and many other kinds of conversion as well.

There are two types of format

Character formats

Transform standard character data into an alternative character form for display or output.

Character formats have the pattern: `$<format-name>w.` *w* is the total width of the output field and the terminating period is mandatory. The *w* is optional and if omitted, a default value will be used.

Numeric formats

Transforms standard numeric data into an alternative numeric form for display or output.

Numeric formats have the pattern: `<format-name>w.d.` *w* is the total width of the output field and *d* is usually the number of decimal places required within the total width of the field.

Note:

The period following the output name is mandatory.

The default numeric format – *w.d* – does not have a rendered name. Both *w* and *d* are optional, if omitted, default values are used.

With all formats, specifying *w* guarantees that *w* characters are always produced – output will be truncated or padded as necessary. As well as giving you access to hundreds of built-in formats, WPS allows you to create your own custom formats with `PROC FORMAT`.

Example format use cases

The following example uses the `$w.` character format to process a string:

```
data _null_;
  d="World Programming";
  put d $5. ;
run;
```

Producing the following output:

```
World
```

In this example, the `w.d` numeric format is used to format a decimal:

```
data _null_;
  d=12345.678;
  put d 8.2;
run;
```

Which produces the following output:

```
12345.68
```

Core formats

Details some of the most widely-used formats, all of which are fundamental to the SAS language.

w.d

Reads in numeric data of length *w* with *d* decimal places.

The format restricts the number of decimal places to be smaller than the format width. If the number is too big to fit in the available space, an attempt is made to express it in scientific notation.

	min	max	default
Variable width	1	32767	1
Decimal digits	0	31	0

Example

```
DATA _null_;
  n = 13.45;
  PUT n = 4.1;
  r = 123456;
  PUT r = 4.1;
RUN;
```

Which produces the following output in the log:

```
n=13.5
r=12E4
```

\$w.

Passes its input string through `untransformed`, retaining trailing blanks.

Truncation occurs if the available output space is too small. It is exactly the same as the `$CHARw.` and `$FW.` formats.

	min	max	default
Variable width	1	32767	*1

Example

This example reads a string and truncates to the first character; truncates the string to the first six characters; finally, as the string is 20 characters long, pads the string to the required 32 characters .

```
DATA _null_;
  s = "World Programming  #";
  PUT s $1.;
  PUT s $6. " *";
  PUT s $32. " *";
RUN;
```

Which produces the following output in the log:

```
W
World *
World Programming  #          *
```

\$CHARw.

Writes its input string, padding to the specified width if its input string is shorter.

The format truncates the output when the number of spaces specified is insufficient to contain the full input string. This format is the same as the `$w.` and `$FW.` formats.

	minimum	maximum	default
Variable width	1	32767	8

Example

This example reads a string and truncates to the first character; truncates the string to the first six characters; finally, as the string is 20 characters long, pads the string to the required 32 characters .

```
DATA _null_;
  s = "World Programming  #";
  PUT s $CHAR1.;
  PUT s $CHAR6. " *";
  PUT s $CHAR32. " *";
RUN;
```

Which produces the following output in the log:

```
W
World *
World Programming  #          *
```

\$Fw.

Passes its input string through untransformed, padding as required to match any specified width.

Truncation occurs if the defined output width is too small. This format is identical to the \$w. and \$CHARw. formats.

	min	max	default
Variable width	1	32767	*1

Example

This example reads a string and truncates to the first character; truncates the string to the first six characters; finally, as the string is 20 characters long, pads the string to the required 32 characters .

```
DATA _null_;
  s = "World Programming  #";
  PUT s $F1.;
  PUT s $F6. " *";
  PUT s $F32. " *";
RUN;
```

Which produces the following output in the log:

```
W
World *
World Programming  #          *
```

\$VARYINGw.

This format can contain a varying number of characters up to the maximim width specified in the format definition.

This character format can be used in procedures where it behaves exactly like \$CHARw. and \$Fw. Within DATA steps, the \$VARYINGw. format can be followed with the name of a variable that contains the actual, runtime-determined width to use. It is limited in length to the format-specified width.

	min	max	default
Variable width	1	32767	*1

Example

```
DATA _null_;
  name="World Programming";
  DO n = 1,2,30,100;
    PUT name $VARYING50. n " ";
  END;
RUN;
```

Which produces the following output in the log:

```
W*
Wo*
World Programming      *
World Programming      *
```

BESTw.

The default numeric format, this attempts to create the optimal numeric representation of its input.

Although a numeric format, it does not require a d modifier, because determining the number of decimal places is part of the work it undertakes itself. This format restricts the requested number of decimal places to be smaller than the format width. If the space available is insufficient to represent the number, an attempt is made to use scientific notation. If the number simply cannot be represented in the available space, asterisks are returned.

	min	max	default
Variable width	1	32767	12
Decimal digits	0	31	0

Example

```
DATA _null_;  
  n=1234.5678;  
  PUT n= BEST1. ;  
  PUT n= BEST2. ;  
  PUT n= BEST3. ;  
  PUT n= BEST4. ;  
  PUT n= BEST6. ;  
  PUT n= BEST10. ;  
run;
```

Which produces the following output in the log:

```
n= *  
n= **  
n=1E3  
n=1235  
n=1234.6  
n=1234.5678
```

Basic character formats

Fundamental character formats for different platforms.

\$ASCIIw.

On an ASCII platform, this format behaves like the \$CHARw. format. On EBCDIC platforms, it converts EBCDIC data to ASCII first.

	min	max	default
Variable width	1	32767	*1

\$BASE64Xw.

This character format converts character data to a base64 encoded form – it may be padded with blanks to the specified number of characters.

This encoding scheme is commonly used when there is a need to encode data in a way that can be transferred over media designed to deal with textual data rather than binary data.

	min	max	default
Variable width	1	32767	*4/3

Example

```
DATA _null_;  
  s="a";  
  PUT s $base64x. "*";  
  PUT s $base64x50. "*";  
RUN;
```

Which produces the following output in the log:

```
YQ==*  
YQ==          *
```

\$BINARYw.

This format converts input character data to a textual representation of its binary value – a string of 1s and 0s.

	min	max	default
Variable width	1	32767	*8

Example

```
DATA _null_;  
  s="a";  
  t="World Programming";  
  PUT s $binary. "*";  
  PUT t $binary. "*";  
RUN;
```

Which produces the following output in the log:

```
01100001*  
0101011101101111011100100110110001100100001000000101000001110010011011  
110110011101110010011000010110110101101101011010010110111001100111*
```

\$BYVALw.

This format is for use by `CALL MODULE`. It is a character format that allows you to pass a single character to a module as an integer.

	min	max	default
Variable width	1	32767	2

\$CSTRw.

This format strips trailing blanks then null-terminates its input character data, and is useful when interfacing with the C-family of programming languages.

If the specified length of the output is less than the length of the input data, truncation occurs, giving precedence to the terminating null. A terminating null is always generated, even if this means overwriting part of the input. If the specified length of the output is longer than the length of the input data plus null, the remaining spaces are undefined.

	min	max	default
Variable width	1	32767	1

Example

```
DATA _null_;  
  s="World Programming";  
  r=put(s, $cstr.);  
  PUT r $hex36. ;  
RUN;
```

Which produces the following output in the log:

```
576F726C642050726F6772616D6D696E00
```

\$EBCDICw.

On an EBCDIC platform, this behaves like the \$CHARw. format. On an ASCII platform, input data is converted to EBCDIC first.

	min	max	default
Variable width	1	32767	*1

\$HEXw.

Converts character data into a hexadecimal form, padding with blanks where relevant.

The output is textual, and it prints the hexadecimal representation of each input character using the characters 0–9A–F, generating two output characters for each input character.

	min	max	default
Variable width	1	32767	*2

Example

```
DATA _null_;  
s="World Programming";  
PUT s $HEX.;  
RUN;
```

Which produces the following output in the log:

```
576F726C642050726F6772616D6D696E67
```

\$MSGCASEw.

Converts character data to the same case as is determined by the MSGCASE system option.

	min	max	default
Variable width	1	32767	1

Example

```
OPTION MSGCASE;  
DATA _null_;  
  s="World Programming";  
  PUT s $MSGCASE17.;  
RUN;  
  
OPTION NOMSGCASE;  
DATA _null_;  
  s="World Programming";  
  PUT s $MSGCASE17.;  
RUN;
```

Which produces the following output in the log:

```
WORLD PROGRAMMING  
...  
World Programming
```

\$OCTALw.

Converts character data into an octal form, padding with blanks where relevant.

The output is textual, and it prints the octal representation of each input character using the characters 0–7, generating three output characters for each input character.

	min	max	default
Variable width	1	32767	*3

Example

```
DATA _null_;  
s="World Programming";  
PUT s $OCTAL.;  
RUN;
```

Which produces the following output in the log:

```
127157162154144040120162157147162141155155151156147
```

\$QUOTEw.

Surrounds character data with double quotation marks, padding with blanks after the final output quote where required.

Output is only generated if there is sufficient space – that is, either both sets of quotation marks are added, or the output will consist of all blanks. Embedded quotes are not doubled up, which means that this format cannot be used to reliably create string literals for the SAS language.

	min	max	default
Variable width	2	32767	+2

Example

```
DATA _null_;  
s="World Programming";  
PUT s $QUOTE40. " *";  
RUN;
```

Which produces the following output in the log:

```
"World Programming" *
```

\$REVERJw.

Converts character data into a visually reversed form without trimming blanks.

	min	max	default
Variable width	1	32767	*1

Example

```
DATA _null_;  
  s = "World Programming  ";  
  PUT s $REVERJ20. "*";  
RUN;
```

Which produces the following output in the log:

```
gnimmargorP dlroW*
```

\$REVERSw.

Converts character data into a visually reversed form, stripping trailing blanks first.

	min	max	default
Variable width	1	32767	*1

Example

```
DATA _null_;  
  s = "World Programming  ";  
  PUT s $REVERJ17.;  
RUN;
```

Which produces the following output in the log:

```
gnimmargorP dlroW
```

\$UPCASEw.

Converts lowercase character data into their uppercase equivalent.

	min	max	default
Variable width	1	32767	*1

Example

```
DATA _null_;  
  s = "World Programming";  
  PUT s $UPCASE.;  
RUN;
```

Which produces the following output in the log:

```
WORLD PROGRAMMING
```

Bidirectional formats

Character formats that deal with data written from left to right and from right to left.

\$BIDIw.

Takes a string in logical order and applies the Unicode bidi algorithm described in <http://www.unicode.org/reports/tr9/>.

	min	max	default
Variable width	1	32767	200

Example

```
DATA _null_;
  LENGTH str $ 16;
  LENGTH logical $ 16;
  logical=UNICODE(
    '\u0063\u0061\u0072\u0020\u0064\u0061\u0065\u006e\u0073\u0020\u0064\u0065\u0068',
    'esc');
  PUT "Logical unicode utf8";
  PUT logical $HEX32.;
  str=PUT(logical, $BIDI16.);
  PUT "bidi";
  PUT str $HEX32.;
run;
```

Which produces the following output in the log:

```
Logical unicode utf8
6361722064616565732064616568
bidi
202020646165657320636172
```

\$LOGVS w.

Takes a string in logical order and returns a string in visual order.

	min	max	default
Variable width	1	32767	200

Example

```
DATA logvs;  
  LENGTH str $ 16;  
  LENGTH logical $ 16;  
  logical=UNICODE(  
    '\u0063\u0061\u0072\u0020\u0064\u0061\u0065\u006e\u0073\u0020\u005b\u005d\u005e',  
    'esc');  
  PUT "Logical unicode utf8";  
  PUT logical $HEX32.;  
  str=PUT(logical, $LOGVS16.);  
  PUT "logvs";  
  PUT str $HEX32.;  
RUN;
```

The logvs dataset contains the encoded output, and the following is output in the log.

```
Logical unicode utf8  
636172206461656E7320D79BD790D7A8  
logvs  
636172206461656E7320D7A8D790D79B
```

\$LOGVSRw.

Reads a string in a right-to-left logical order and returns it in a visual order.

	min	max	default
Variable width	1	32767	200

Example

```
DATA logvsr;  
  LENGTH str $ 16;  
  LENGTH logicalr $ 16;  
  logicalr=UNICODE(  
    '\u005b\u005d\u005e\u0020\u0063\u0061\u0072\u0020\u0064\u0065\u0061\u006e\u0073',  
    'esc');  
  PUT "Logical RTL unicode utf8";  
  PUT logicalr $HEX32.;  
  str=PUT(logicalr, $LOGVSR16.);  
  PUT "logvsr";  
  PUT str $HEX32.;  
RUN;
```

The logvsr dataset contains the encoded output, and the following is output in the log.

```
Logical RTL unicode utf8  
D79BD790D7A820636172206461656E73  
logvsr  
636172206461656E7320D7A8D790D79B
```

\$VSLOGw.

Takes a string in visual order and returns it in logical order.

	min	max	default
Variable width	1	32767	200

Example

```
DATA _null_;
  LENGTH str $ 16;
  LENGTH visual $ 16;
  visual=UNICODE(
    '\u0063\u0061\u0072\u0020\u0064\u0061\u0065\u006e\u0073\u0020\u005e\u005d\u005b',
    'esc');
  PUT "Visual unicode utf8";
  PUT visual $HEX32.;
  str=put(visual, $VSLOGR16.);
  PUT "vslogr";
  PUT str $HEX32.;
RUN;
```

Which produces the following output in the log:

```
Visual unicode utf8
6361722064616565E7320D7A8D790D79B
vslog
6361722064616565E7320D79BD790D7A8
```

\$VSLOGRw.

Takes a string in visual order and returns it in logical right-to-left order.

	min	max	default
Variable width	1	32767	200

Example

```
data _null_;
  length str $ 16;
  length visual $ 16;
  visual=unicode(
    /* car means RAC */
    /* car means resh alef kaf */
    '\u0063\u0061\u0072\u0020\u0064\u0061\u0065\u006e\u0073\u0020\u005e\u005d\u005db',
    'esc');
  put "Visual unicode utf8";
  put visual $hex32.;
  str=put( visual, $vslogr16. );
  put "vslogr";
  put str $hex32.;
run;
```

Which produces the following output in the log:

```
Visual unicode utf8
636172206461656E7320D7A8D790D79B
vslogr
D79BD790D7A820636172206461656E73
```

Unicode formats

Character formats for different variants of the Unicode encoding.

\$UCS2Bw.

Converts character data to big-endian, 16-bit UCS2 Unicode encoding.

The input string is converted from session encoding into Unicode, then for each Unicode character, two bytes are written out in big-endian order, yielding the big-endian UCS2 encoding of that character. UCS2 always produces two bytes of output for each character.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;
  s="World Programming";
  r=PUT(s, $UCS2B34.);
  PUT r $HEX.;
RUN;
```

Which produces the following output in the log:

```
0057006F0072006C0064002000500072006F006700720061006D006D0069006E0067
```

\$UCS2BEw.

Converts character strings in big-endian, 16-bit UCS2 Unicode encoding to the same strings in the session encoding. It performs the reverse of \$UCS2B.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;  
  s="World Programming";  
  t=put(s, $UCS2B34.);  
  PUT t;  
  u=PUT(t, $UCS2BE17.);  
  PUT u;  
RUN;
```

Which produces the following output in the log:

```
W o r l d   P r o g r a m m i n g  
World Programming
```

\$UCS2Lw.

Converts character data to little-endian, 16-bit UCS2 Unicode encoding.

The input string is converted from session encoding into Unicode, then for each Unicode character, two bytes are written out in little-endian order, yielding the little-endian UCS2 encoding of that character. UCS2 always produces two bytes of output for each character.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;  
  s = "World Programming";  
  t = PUT(s, $UCS2L34.);  
  PUT t $HEX.;  
RUN;
```

Which produces the following output in the log:

```
57006F0072006C0064002000500072006F006700720061006D006D0069006E006700
```

\$UCS2LEw.

This format converts character strings in little-endian, 16-bit UCS2 Unicode encoding to the same strings in the session encoding. It performs the reverse of \$UCS2L.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;
  a = "World Programming";
  b = PUT(a, $UCS2L34.);
  PUT b;
  t = PUT(b, $UCS2LE17.);
  PUT t;
RUN;
```

Which produces the following output in the log:

```
W o r l d   P r o g r a m m i n g
World Programming
```

\$UCS2Xw.

This format converts character data to a 16-bit UCS2 Unicode machine-endian encoding.

This format operates exactly the same as \$UCS2B or \$UCS2L, as appropriate to the endianness of the executing machine.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCX2X34.);
  PUT t $hex.;
RUN;
```

Which produces the following output in the log (if executed on an x86 machine):

```
57006F0072006C0064002000500072006F006700720061006D006D0069006E006700
```

\$UCS2XEw.

This format converts character data from a 16-bit UCS2 Unicode machine-endian encoding to a session-encoded form.

It operates exactly the same as \$UCS2BE or \$UCS2LE as is appropriate to the endianness of the executing machine.

	min	max	default
Variable width	2	32767	8

Example

```
DATA _null_;  
  s = "World Programming";  
  t = PUT(s, $UCS2X34.);  
  PUT t;  
  u = PUT(t, $UCS2XE17.);  
  PUT u;  
RUN;
```

Which produces the following output in the log (if executed on an x86 machine):

```
W o r l d   P r o g r a m m i n g  
World Programming
```

\$UCS4Bw.

This format converts character data to big-endian, 32-bit UCS4 Unicode encoding.

The input string is converted from session encoding into Unicode, then for each Unicode character, four bytes are written out in big-endian order, yielding the big-endian UCS4 encoding of that character. UCS4 always produces four bytes of output for each character.

	min	max	default
Variable width	4	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4B68.);
  PUT t $HEX.;
RUN;
```

Which produces the following output in the log:

```
0000005700000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067
```

\$UCS4BEw.

This format converts character strings in big-endian, 32-bit UCS4 Unicode encoding to the same strings in the session encoding. It performs the reverse of \$UCS4B.

	min	max	default
Variable width	4	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4B68.);
  PUT t $HEX.;
  u = PUT(t, $UCS4BE34.);
  PUT u;
RUN;
```

Which produces the following output in the log:

```
0000005700000006F000000720000006C00000064000000200000005000000072000000
6F0000006700000072000000610000006D0000006D000000690000006E00000067
W o r l d   P r o g r a m m i n g
```

\$UCS4Lw.

This format converts character data to little-endian, 32-bit UCS4 Unicode encoding.

The input string is converted from session encoding into Unicode, then for each Unicode character, four bytes are written out in little-endian order, yielding the little-endian UCS4 encoding of that character. UCS4 always produces four bytes of output for each character.

	min	max	default
Variable width	4	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4L68.);
  PUT t $HEX.;
RUN;
```

Which produces the following output in the log:

```
570000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067000000
```

\$UCS4LEw.

This format converts character strings in big-endian, 32-bit UCS4 Unicode encoding to the same strings in the session encoding. It performs the reverse of \$UCS4L.

	min	max	default
Variable width	4	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4L68.);
  PUT t $HEX.;
  u = PUT(t, $UCS4LE34.);
  PUT u;
RUN;
```

Which produces the following output in the log:

```
570000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067000000
W o r l d   P r o g r a m m i n g
```

\$UCS4Xw.

This format converts character data to a 32-bit UCS4 Unicode machine-endian encoding. It operates exactly the same as \$UCS4B or \$UCS4L, as appropriate to the endianness of the executing machine.

	min	max	default
Variable width	4	32767	8

Example

```
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4X68.);
  PUT t $HEX.;
RUN;
```

Which produces the following output in the log (if executed on an x86 machine):

```
570000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067000000
```

\$UCS4XEw.

This format converts character data from a 32-bit UCS4 Unicode machine-endian encoding to a session-encoded form. It operates exactly the same as \$UCS4BE or \$UCS4LE as is appropriate to the endianness of the executing machine.

Example

```
OPTIONS ENCODING='UTF-8';
DATA _null_;
  s = "World Programming";
  t = PUT(s, $UCS4X68.);
  PUT t $HEX.;
  u = PUT(t, $UCS4XE34.);
  PUT u;
RUN;
```

Which produces the following output in the log (if executed on an x86 machine):

```
570000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067000000
W o r l d   P r o g r a m m i n g
```

\$UESCw.

This format converts all but the 0–9, A–Z, a–z and space characters (in session encoding) to Unicode universal character names in the \uXXXX notation.

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;  
  s = "#World Programming&";  
  t = PUT(s, $UESC30.);  
  PUT t ;  
RUN;
```

Which produces the following output in the log:

```
\u0023World Programming\u0026
```

\$UESCEw.

This format decodes universal character names in the \uXXXX format, leaving other characters alone.

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;  
  s = "\u0023World Programming\u0026";  
  t = PUT(s, $UESCE30.);  
  PUT t ;  
RUN;
```

Which produces the following output in the log:

```
#World Programming&
```

\$UNCRw.

This format converts all but 0–9, A–Z, a–z and space to the Unicode numeric character reference format (the &#dddd notation). This notation is described in <http://www.w3.org/TR/html4/charset.html#h-5.3.1> [↗](#).

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;  
  s = "#World Programming";  
  t = PUT(s, $UNCR40.);  
  PUT t ;  
RUN;
```

Which produces the following output in the log:

```
&#00035;World Programming&#00038;
```

\$UNCREw.

This format decodes numeric character references in the &#dddd; notation, leaving other characters alone.

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;  
  s = "&#00035;World Programming&#00038;";  
  t = PUT(s, $UNCRE40.);  
  PUT t ;  
RUN;
```

Which produces the following output in the log:

```
#World Programming&
```

\$UPARENw.

This format converts each character in a string to the format <uXXXX> where XXXX is the Unicode code point for the character in hexadecimal.

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;  
  s = "World Programming";  
  t = PUT(s, $UPAREN200.);  
  PUT t ;  
RUN;
```

Which produces the following output in the log:

```
<u0057><u006F><u0072><u006C><u0064><u0020><u0050><u0072><u006F><u0067>
<u0072><u0061><u006D><u006D><u0069><u006E><u0067>
```

\$UPARENEw.

This format decodes a string of UPAREN-encoded sequences.

	min	max	default
Variable width	1	32767	8

Example

```
DATA _null_;
  s = "<u0057><u006F><u0072><u006C><u0064><u0020><u0050><u0072><u006F>
      <u0067><u0072><u0061><u006D><u006D><u0069><u006E><u0067>";
  t = PUT(s, $UPARENE20.);
  PUT t ;
RUN;
```

Which produces the following output in the log:

```
World Programming
```

\$UTF8Xw.

This format converts a string from session encoding to UTF-8.

	min	max	default
Variable width	1	32767	8

\$UTF8XEw.

This format converts character data from UTF-8 to session encoding. The input to this format is always in UTF-8, regardless of the session encoding. It is then converted to session encoding and output.

	min	max	default
Variable width	1	32767	8

Simple numeric formats

Fundamental formats for numeric data.

BESTX w .

This numeric format is the same as the F w . format, but with a different default width.

	min	max	default
Variable width	1	32	12

Example

```
DATA _null;  
  d=0.7;  
  e=1.5;  
  PUT d bestx. " "  
  PUT e bestx. " "  
  PUT d bestx8.1 " "  
  PUT d bestx8.2 " "  
RUN;
```

Which produces the following output in the log:

```
      1*  
      2*  
0.7*  
0.70*
```

BINARY w .

This format converts numeric data to a binary representation – a textual representation of a binary number consisting of a string of 1s and 0s.

The format width determines whether the number is considered an integer or a double. For widths of 59 or more, the output consists of the bits of the double floating point number – 'digits' is ignored. For widths of less than 59, the number is considered to be an integer and the 'digits' field is a scaling power of 10.

	min	max	default
Variable width	1	64	8

Example

```
DATA _null_;
  s=1;
  t=8;
  PUT s binary. " ";
  PUT s binary20. " ";
  PUT t binary. " ";
  PUT t binary20. " ";
RUN;
```

Which produces the following output in the log:

```
00000001*
000000000000000000000001*
00001000*
00000000000000000001000*
```

COMMAw.d

This format decorates numeric data by inserting commas every three digits, counting from the right.

When using this format, the decimal point is represented by a period. Commas are only inserted if the specified output length is sufficient to accommodate them.

	min	max	default
Variable width	1	32	6

Example

```
DATA _null_;
  s = 1234567.89;
  PUT s COMMA.3 " ";
  PUT s COMMA12.3 " ";
  PUT s COMMA20.3 " ";
RUN;
```

Which produces the following output in the log:

```
1.23E6*
1234567.890*
  1,234,567.890*
```

COMMAXw.d

This format behaves like `COMMAw.d` except that it inserts periods, not commas. The decimal point is represented by a comma.

	min	max	default
Variable width	1	32	6

Example

```
DATA _null_;  
  s=1234567.89;  
  PUT s COMMAX.3;  
  PUT s COMMAX12.3;  
  PUT s COMMAX20.3;  
RUN;
```

Which produces the following output in the log:

```
1,23E6  
1234567,890  
1.234.567,890
```

Dw.d

This format converts numeric values into a form where, when output, the decimal points are likely to line up.

	min	max	default
Variable width	1	32	12

Example

```
DATA _null_;  
  s=1234567.89;  
  t=22.323;  
  PUT s D20.3;  
  PUT t D20.3;  
RUN;
```

Which produces the following output in the log:

```
1234567.8900000000  
22.3230000000
```

DOLLARw.d

This format prepends a dollar sign to a numeric value.

In addition to the dollar sign, commas are inserted between sets of three digits to improve readability (just like COMMAw.d). These actions are only taken if there is sufficient room for the output. This format is not locale-sensitive, a dollar character is always output.

	min	max	default
Variable width	2	32	6

Example

```
DATA _null_;  
  s=1096543.123;  
  PUT s DOLLAR20.3;  
RUN;
```

Which produces the following output in the log:

```
$1,096,543.123
```

DOLLARXw.d

This format prepends a dollar sign to a numeric value.

In addition to the dollar sign, periods between sets of three digits to improve readability (just like COMMAXw.d). These actions are only taken if there is sufficient room for the output.

	min	max	default
Variable width	2	32	6

Example

```
DATA _null_;  
  s=1096543.123;  
  PUT s DOLLARX20.3;  
RUN;
```

Which produces the following output in the log:

```
$1.096.543,123
```

Ew.

This format converts its numeric input to scientific notation.

	min	max	default
Variable width	7	32	12

Example

```
DATA _null_;  
  s=1096543.123;  
  PUT s E20.;  
RUN;
```

Which produces the following output in the log:

```
1.0965431230000E+06
```

EUROW.d

This format prepends an 'E' character to a numeric value to represent the Euro (€) character.

In addition to the 'E' character, the thousand separators are formatted as a comma, and the decimal separator as a period.

	min	max	default
Variable width	2	32	6

Example

```
DATA _null_;  
  s=1096543.123;  
  PUT s EURO20.3;  
RUN;
```

Which produces the following output in the log:

```
E1,096,543.123
```

EUROXw.d

This format prepends an 'E' character to a numeric value to represent the Euro (€) character.

In addition to the 'E' character, the thousand separators are formatted as a period, and the decimal separator as a comma.

	min	max	default
Variable width	2	32	6

Example

```
DATA _null_;  
  s=1096543.123;  
  PUT s EUROX20.3;  
RUN;
```

Which produces the following output in the log:

```
E1.096.543,123
```

Fw.d

F is an alternative name for the standard numeric format.

In practice, it is unusual to specify the format fully – for example, 7.4 is generally used, rather than F7.4. The number of decimal places is restricted to be smaller than the format width.

	min	max	default
Variable width	1	32	1

Example

```
DATA _null_;  
  s = 123.456;  
  PUT s F7.3;  
  PUT s 7.3;  
RUN;
```

Which produces the following output in the log:

```
123.456  
123.456
```

FLOATw.d

This format converts numeric data to a 4-byte single precision floating point value, outputting the result as a binary value. The decimal part is a power of 10 by which the value part is multiplied.

	min	max	default
Variable width	4	4	4

Example

```
DATA _null_;
  s = 1;
  r = PUT(s, FLOAT.);
  PUT r $HEX.;
RUN;
```

Which produces the following hex representation of the format in the log:

```
0000803F
```

FRACTw.

This format finds the nearest fraction to the input numeric value – it uses the method of continued fractions.

	min	max	default
Variable width	4	32	10

Example

```
DATA _null_;
  s = 1.333;
  PUT s FRACT. " *";
RUN;
```

Which produces the following output in the log:

```
1+333/1000*
```

HEXw.

This format converts a numeric value to a hexadecimal representation, outputting the result in a textual form consisting of the characters 0–9, A–F.

If the width is 16, the floating point representation is output. For lesser widths, the output is converted to an integer and the hex representation of that output.

	min	max	default
Variable width	1	16	8

Example

```
DATA _null_;  
  s = 15;  
  PUT s HEX.;  
RUN;
```

Which produces the following output in the log:

```
0000000F
```

IBw.d

This format converts a number into a native-endian integer in binary format.

On Intel platforms, it converts the number 42 into 2A 00 00 00, while on big-endian platforms, the output would be 00 00 00 2A. The number of digits represents a scaling power of 10.

A double floating point value is converted into a signed 64-bit integer. Input values outside the range of a signed, 64-bit integer take the largest positive or negative value as appropriate. As many of the bytes of the integer are output as requested by the length, truncating on the most significant end if necessary.

Missing input values are output as 0.

	min	max	default
Variable width	1	8	4

Example

```
DATA _null_;  
  n = 42;  
  s = PUT(n, IB.);  
  PUT s $HEX.;  
RUN;
```

Which produces the following hex representation of the format output in the log (on Intel platforms):

```
2A000000
```

IBRw.d

This format converts a number into a little-endian integer in binary format.

when converting, the number of digits represents a scaling power of 10; for example the number 42 is converted to 2A 00 00 00.

	min	max	default
Variable width	1	8	4

Example

```
DATA _null_;  
  n = 42;  
  s = PUT(n,IBR.);  
  PUT s $HEX.;  
RUN;
```

Which produces the following output in the log:

```
2A000000
```

IEEEw.d

This format converts numeric input to an IEEE floating point representation.

If present, the number of digits is a scaling power of 10.

If *w* is 1–4, the result is a single-precision IEEE floating point number. On IEEE platforms, for widths of 1–4, the number is first converted to single precision if possible, and then the bytes of the float are output, truncating at the least significant end of the fraction if necessary.

If *w* is 5–8, the result is a double-precision IEEE floating point number. On IEEE platforms, for widths of 5–8, this format outputs a (potentially truncated) double precision floating point number in binary format, copying bytes from memory to the output. If necessary, truncation occurs at the least significant end of the fraction of the number, so some precision is lost.

On z/OS, a similar process occurs, except that the input number is first converted from IBM hex floating point to IEEE.

	min	max	default
Variable width	1	8	8

Example

```
DATA _null_;
  LENGTH s1 $ 8 s2 $ 16;
  DO n = 1, 1234567.95, ., .A, 1E60;
    PUT n @;
    DO w = 3, 4, 7;
      DO d = 0, 4;
        s1 = PUTN(n, 'IEEE', w, d);
        s2 = PUTC(s1, '$HEX', w*2);
        PUT @20 ': IEEE' w +(-1) '.' d s2;
      END;
    END;
  END;
RUN;
```

Which produces the following output in the log:

```
1                : ieee3.0 3F8000
                  : ieee3.4 461C40
                  : ieee4.0 3F800000
                  : ieee4.4 461C4000
                  : ieee7.0 3FF000000000000
                  : ieee7.4 40C388000000000
1234567.95       : ieee3.0 4996B4
                  : ieee3.4 5037F7
                  : ieee4.0 4996B43F
                  : ieee4.4 5037F707
                  : ieee7.0 4132D687F33333
                  : ieee7.4 4206FEE0F46000
.                : ieee3.0 FFE880
                  : ieee3.4 FFE880
                  : ieee4.0 FFE88000
                  : ieee4.4 FFE88000
                  : ieee7.0 FFFFD1000000000
                  : ieee7.4 FFFFD1000000000
A                : ieee3.0 FFE880
                  : ieee3.4 FFE880
                  : ieee4.0 FFE88000
                  : ieee4.4 FFE88000
                  : ieee7.0 FFFFBE000000000
                  : ieee7.4 FFFFBE000000000
1E60             : ieee3.0 FFE880
                  : ieee3.4 FFE880
                  : ieee4.0 FFE88000
                  : ieee4.4 FFE88000
                  : ieee7.0 4C63E9E4E4C2F3
                  : ieee7.4 4D384F03E93FF9
```

NEGPAREN*w.d*

This format places commas after every three digits to improve readability.

If the value is negative, the minus sign is replaced by surrounding parentheses as is the style in the finance industry. For correct operation, sufficient space must be specified in the width of the output to include the parentheses.

	min	max	default
Variable width	1	32	6

Example

The following example formats the two supplied negative numbers, but does not attempt to format the number where the default length is insufficient.

```
DATA _null_;
  d = -10;
  e = -1000;
  PUT d NEGPARAN. " ";
  PUT e NEGPARAN. " ";
  PUT d NEGPARAN10.2 " ";
  PUT e NEGPARAN10.2 " ";
RUN;
```

Which produces the following output in the log:

```
(10) *
-1,000 *
(10.00) *
(1,000.00) *
```

NUMX*w.d*

This format converts a decimal point in a numeric variable into a comma.

This format is the same as the regular blank numeric format, except that it uses a comma instead of a decimal point.

	min	max	default
Variable width	1	32	12

Example

```
DATA _null_;
  s = 123456.789;
  t = PUT(s, NUMX12.4);
  PUT t;
RUN;
```

Which produces the following output in the log:

```
123456,7890
```

OCTAL*w.d*

This format converts a numeric input to its octal representation.

If present, the number of digits (d) is a scaling power of 10; the input is multiplied by the scaling factor, and the format applied to the output. The formatted output will be truncated if the width is too narrow to contain the converted value, any truncation occurs on the left.

	min	max	default
Variable width	1	24	3

Example

The following outputs the default octal value for the input value of 8. This value is then scaled by 10^2 to 800 decimal (1440 octal), and by 10^3 to generate a result that is too large to fit the format width.

```
DATA _null_;  
  d = 8;  
  PUT d OCTAL. " ";  
  PUT d OCTAL4.2 " ";  
  PUT d OCTAL4.3 " ";  
RUN;
```

Which produces the following output in the log:

```
010*  
1440*  
7500*
```

ODDSR*w.d*

This format transforms its numeric argument into an odds ratio.

	min	max	default
Variable width	2	32	8

Example

```
DATA _null;
  d = 1.34;
  e = -1.34;
  PUT d ODDSR. " ";
  PUT e ODDSR. " ";
  PUT d ODDSR5.2 " ";
  PUT e ODDSR5.2 " ";
RUN;
```

Which produces the following output in the log:

```
1.340*
<0.001*
1.34*
<0.01*
```

PDw.d

This format converts numeric input arguments to a packed decimal form where two digits are encoded into one byte with one digit per four-byte nibble.

The representation is platform-dependent – its mainframe representation differs from its representation on other architectures. If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	16	1

Example

```
DATA _null_;
  s = 12.0;
  r = PUT(s, PD4.2);
  PUT r $HEX.;
RUN;
```

Which produces the following output in the log (on Intel platforms):

```
00001200
```

PERCENTw.d

This format converts its numeric input into a percentage – for example, an input of 0.5 would produce an output of 50%.

If present, the number of digits (d) is a scaling power of 10. Negative numbers are surrounded with parentheses and the format leaves space in the output to allow for this.

	min	max	default
Variable width	4	32	6

Example

```
DATA _null;
  d = 0.7;
  e = 1.5;
  f = -2.1;
  PUT d PERCENT. " ";
  PUT e PERCENT. " ";
  PUT d PERCENT8.1 " ";
  PUT d PERCENT8.2 " ";
  PUT f PERCENT. " ";
RUN;
```

Which produces the following output in the log:

```
70% *
150% *
70.0% *
70.00% *
(210%) *
```

PERCENTNw.d

This format converts its numeric input into a percentage – for example, an input of 0.25 would produce an output of 25%.

If present, the number of digits (d) is a scaling power of 10. Negative numbers have a minus sign in front of them and the format leaves space in the output to allow for this. Additionally, PERCENTN also leaves a space at the end, similar to PERCENT.

	min	max	default
Variable width	4	32	6

PIBw.d

This format converts its numeric input into a positive integer binary representation – the output is binary, not textual.

For negative inputs, the output is set to a maximum binary value. The number of digits (d), if present, is a scaling power of 10.

	min	max	default
Variable width	1	8	1

Example

```
DATA _null_;
  LENGTH s1 $ 8 s2 $ 16;
  DO n = -10, 1, 1234567.95, ., .A, 1E60;
    PUT n @;
    DO w = 3, 4, 7;
      DO d = 0, 4;
        s1 = PUTN(n, 'PIB', w, d);
        s2 = PUTC(s1, '$HEX', w*2);
        PUT @20 ': PIB' w +(-1) '.' d s2;
      END;
    END;
  END;
RUN;
```

Which produces the following output in the log:

```
-10          : pib3.0 FFFFFFFF
              : pib3.4 FFFFFFFF
              : pib4.0 FFFFFFFF
              : pib4.4 FFFFFFFF
              : pib7.0 FFFFFFFF
              : pib7.4 FFFFFFFF
1            : pib3.0 010000
              : pib3.4 102700
              : pib4.0 01000000
              : pib4.4 10270000
              : pib7.0 01000000000000
              : pib7.4 10270000000000
1234567.95   : pib3.0 88D612
              : pib3.4 FFFFFFFF
              : pib4.0 88D61200
              : pib4.4 FFFFFFFF
              : pib7.0 88D61200000000
              : pib7.4 8C1EDCDF020000
.            : pib3.0 000000
              : pib3.4 000000
              : pib4.0 00000000
              : pib4.4 00000000
              : pib7.0 00000000000000
              : pib7.4 00000000000000
A            : pib3.0 000000
              : pib3.4 000000
              : pib4.0 00000000
              : pib4.4 00000000
              : pib7.0 00000000000000
              : pib7.4 00000000000000
1E60         : pib3.0 FFFFFFFF
              : pib3.4 FFFFFFFF
              : pib4.0 FFFFFFFF
              : pib4.4 FFFFFFFF
              : pib7.0 FFFFFFFF
              : pib7.4 FFFFFFFF
```

PIBR $w.d$

This format converts its numeric input into a positive integer binary representation, guaranteeing that its output will be in a little-endian format.

For negative inputs, the output is set to a maximum binary value. If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	8	1

Example

```
DATA _null_;
  LENGTH s1 $ 8 s2 $ 16;
  DO n = -10, 1, 1234567.95, ., .A, 1E60;
    PUT n @;
  DO w = 3, 4, 7;
    DO d = 0, 4;
      s1 = PUTN(n, 'PIBR', w, d);
      s2 = PUTC(s1, '$HEX', w*2);
      PUT @20 ': PIBR' w +(-1) '.' d s2;
    END;
  END;
END;
RUN;
```

Which produces the following output in the log:

```
-10          : pibr3.0 FFFFFFFF
              : pibr3.4 FFFFFFFF
              : pibr4.0 FFFFFFFF
              : pibr4.4 FFFFFFFF
              : pibr7.0 FFFFFFFF
              : pibr7.4 FFFFFFFF
1            : pibr3.0 010000
              : pibr3.4 102700
              : pibr4.0 01000000
              : pibr4.4 10270000
              : pibr7.0 01000000000000
              : pibr7.4 10270000000000
1234567.95   : pibr3.0 88D612
              : pibr3.4 FFFFFFFF
              : pibr4.0 88D61200
              : pibr4.4 FFFFFFFF
              : pibr7.0 88D61200000000
              : pibr7.4 8C1EDCDF020000
.            : pibr3.0 000000
              : pibr3.4 000000
              : pibr4.0 00000000
              : pibr4.4 00000000
              : pibr7.0 00000000000000
              : pibr7.4 00000000000000
A            : pibr3.0 000000
              : pibr3.4 000000
              : pibr4.0 00000000
```

```

: pibr4.4 00000000
: pibr7.0 0000000000000000
: pibr7.4 0000000000000000
1E60
: pibr3.0 FFFFFFFF
: pibr3.4 FFFFFFFF
: pibr4.0 FFFFFFFF
: pibr4.4 FFFFFFFF
: pibr7.0 FFFFFFFF
: pibr7.4 FFFFFFFF

```

PKw.d

This format converts its numeric argument into a machine-independent packed decimal form.

When converting, two digits are encoded into one byte with one digit per 4-byte nibble. If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	16	1

Example

```

DATA _null_;
  LENGTH s1 $ 8 s2 $ 16;
  DO n = -10, 1, 1234567.95, ., .A, 1E60;
    PUT n @;
    DO w = 3, 4, 7;
      DO d = 0, 4;
        s1 = PUTN(n, 'PK', w, d);
        s2 = PUTC(s1, '$HEX', w*2);
        PUT @20 ': IEEE' w + (-1) '.' d s2;
      END;
    END;
  END;
  RUN;

```

Which produces the following output in the log:

```

-10
: ieee3.0 000010
: ieee3.4 100000
: ieee4.0 00000010
: ieee4.4 00100000
: ieee7.0 000000000000010
: ieee7.4 00000000100000
1
: ieee3.0 000001
: ieee3.4 010000
: ieee4.0 00000001
: ieee4.4 00010000
: ieee7.0 000000000000001
: ieee7.4 00000000010000
1234567.95
: ieee3.0 999999
: ieee3.4 999999
: ieee4.0 01234568
: ieee4.4 99999999

```

```

: ieee7.0 00000001234568
: ieee7.4 00012345679500
.
: ieee3.0 000000
: ieee3.4 000000
: ieee4.0 00000000
: ieee4.4 00000000
: ieee7.0 00000000000000
: ieee7.4 00000000000000
A
: ieee3.0 000000
: ieee3.4 000000
: ieee4.0 00000000
: ieee4.4 00000000
: ieee7.0 00000000000000
: ieee7.4 00000000000000
1E60
: ieee3.0 999999
: ieee3.4 999999
: ieee4.0 99999999
: ieee4.4 99999999
: ieee7.0 99999999999999
: ieee7.4 99999999999999

```

PVALUE $w.d$

This format converts probabilities (especially small ones) into a form in which the specified number of digits determines the smallest number it can output.

Output values less than this minimum (including negative numbers) are rendered with the < symbol. If the number of decimal points is 2, then the smallest number it can render is 0.01. If 3, then it is 0.001, and so on.

	min	max	default
Variable width	3	32	6

Example

```

DATA _null_;
  d = 0.002;
  d1 = 0.0002;
  d2= 0.00003;
  PUT d PVALUE7.2;
  PUT d1 PVALUE7.3;
  PUT d2 PVALUE7.4;
RUN;

```

Which produces the following output in the log:

```

<.01
<.001
<.0001

```

RBw.d

This format simply outputs the requested number of bytes of the double floating point in memory, truncating so as to lose precision from the fraction if necessary.

If present, digits represent a scaling power of 10. There is no conversion to single precision for narrow widths. RB stands for 'real binary'.

	min	max	default
Variable width	2	8	4

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
do n = -10, 1, 1234567.95, ., .A, 1E60;
  put n @;
  do w = 3, 4, 7;
    do d = 0, 4;
      s1 = putn(n, 'rb', w, d);
      s2 = putc(s1, '$hex', w*2);
      put @20 ': rb' w +(-1) '.' d s2;
    end;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : rb3.0 0024C0
              : rb3.4 6AF8C0
              : rb4.0 000024C0
              : rb4.4 006AF8C0
              : rb7.0 000000000024C0
              : rb7.4 000000006AF8C0
1            : rb3.0 00F03F
              : rb3.4 88C340
              : rb4.0 0000F03F
              : rb4.4 0088C340
              : rb7.0 0000000000F03F
              : rb7.4 0000000088C340
1234567.95   : rb3.0 D63241
              : rb3.4 FE0642
              : rb4.0 87D63241
              : rb4.4 E0FE0642
              : rb7.0 3333F387D63241
              : rb7.4 0060F4E0FE0642
.            : rb3.0 D1FFFF
              : rb3.4 D1FFFF
              : rb4.0 00D1FFFF
              : rb4.4 00D1FFFF
              : rb7.0 00000000D1FFFF
              : rb7.4 00000000D1FFFF
A            : rb3.0 BEFFFF
              : rb3.4 BEFFFF
              : rb4.0 00BEFFFF
```

```

1E60      : rb4.4 00BEFFFF
           : rb7.0 00000000BEFFFF
           : rb7.4 00000000BEFFFF
           : rb3.0 E9634C
           : rb3.4 4F384D
           : rb4.0 E4E9634C
           : rb4.4 034F384D
           : rb7.0 F3C2E4E4E9634C
           : rb7.4 F93FE9034F384D

```

ROMANw.

This format converts its numeric input to a Roman numeral form, after first converting into an integer.

	min	max	default
Variable width	2	32	6

Example

```

data _null_;
d=50;
d1 = 121;
put d roman. "*";
put d1 roman. "*";
run;

```

Which produces the following output in the log:

```

L      *
CXXI   *

```

S370FFw.d

First, this format performs a conversion identical to that of the F format, the standard numeric format.

If on a non-z/OS machine, it converts that into open edition 1047 EBCDIC. It has no further effect on z/OS machines, which are native EBCDIC. The output of this format is always in EBCDIC.

	min	max	default
Variable width	1	32	12

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
d=0;
n=10;
put n @;
s1 = putn(n, 's370ff', w, d);
s2 = putc(s1, '$hex', w*2);
put @20 ': s370ff' w +(-1) '.' d s2;
run;
```

Which produces the following output in the log:

```
10                : s370ff5.0 404040F1F0
```

S370FIBw.d

This format converts a number into a big-endian integer in binary format.

If present, the number of digits (d) is a scaling power of 10. A double floating point value is converted into a signed 64-bit integer. Input values outside the range of a signed, 64-bit integer take the largest positive or negative value as appropriate. As many of the bytes of the integer are output as requested by the length, truncating on the most significant end if necessary.

Missing input values are output as 0.

On an IBM mainframe, this format is identical to the IBw.d format.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;
r=42;
s=put(r, s370fib4.);
put s $hex8.;
run;
```

Which produces the following output in the log:

```
0000002A
```

S370FIBUw.d

This format transforms numeric data, taking its absolute value and converting it into big-endian positive integer binary IBM mainframe format.

If present, the number of digits (d) is a scaling power of 10. A double floating point value is converted into a signed 64-bit integer. Input values outside the range of a signed, 64-bit integer take the largest positive or negative value as appropriate. As many of the bytes of the integer are output as requested by the length, truncating on the most significant end if necessary.

Missing input values are output as 0, and negative input values are first converted to their absolute value.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;  
x=-42;  
y=put(x,s370fib4.);  
put y $hex8.;  
run;
```

Which produces the following output in the log:

```
0000002A
```

S370FPDw.d

This format converts numeric data into z/OS packed decimal format where two digits are encoded into one byte with one digit per 4-byte nibble.

If present, the number of digits (d) is a scaling power of 10. This representation of this format is machine-independent, as it always returns the IBM mainframe format in which the sign is signified by the terminating nibble.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fpd',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fpd' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fpd5.0 000000010D
-10          : s370fpd5.1 000000100D
 10          : s370fpd5.0 000000010C
 10          : s370fpd5.1 000000100C
```

S370FPDUw.d

This format transforms its numeric input, taking its absolute value and converting it into z/OS unsigned packed decimal format.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fpdu',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fpdu' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fpdu5.0 000000010F
-10          : s370fpdu5.1 000000100F
 10          : s370fpdu5.0 000000010F
 10          : s370fpdu5.1 000000100F
```

S370FPiB*w.d*

This format converts numeric data into z/OS big-endian positive integer binary format.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fpib',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fpib' w +(-1) ' .' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fpib5.0 FFFFFFFFFF
-10          : s370fpib5.1 FFFFFFFFFF
 10          : s370fpib5.0 000000000A
 10          : s370fpib5.1 0000000064
```

S370FRB*w.d*

This format converts a native representation of floating point numeric data into a z/OS big-endian binary representation.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	2	8	4

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370frb',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370frb' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370frb5.0 C1A0000000
-10          : s370frb5.1 C264000000
10           : s370frb5.0 41A0000000
10           : s370frb5.1 4264000000
```

S370FZDw.d

This format converts numeric data into z/OS zoned decimal format with one digit (or character) per byte.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fzd',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fzd' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fzd5.0 F0F0F0F1D0
-10          : s370fzd5.1 F0F0F1F0D0
10           : s370fzd5.0 F0F0F0F1C0
10           : s370fzd5.1 F0F0F1F0C0
```

S370FZDLw.d

This format converts numeric data into z/OS zoned decimal format with a sign nibble at the beginning.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fzdl',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fzdl' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fzdl5.0 D0F0F0F1F0
-10          : s370fzdl5.1 D0F0F1F0F0
10           : s370fzdl5.0 C0F0F0F1F0
10           : s370fzdl5.1 C0F0F1F0F0
```

S370FZDSw.d

This format converts numeric data into z/OS zoned decimal format with a sign byte at the beginning.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	2	32	8

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fzds',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fzds' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fzds5.0 60F0F0F1F0
-10          : s370fzds5.1 60F0F1F0F0
10           : s370fzds5.0 4EF0F0F1F0
10           : s370fzds5.1 4EF0F1F0F0
```

S370FZDTw.d

This format converts numeric data into z/OS zoned decimal format with a sign byte at the end.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	2	32	8

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fzdt',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fzdt' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fzdt5.0 F0F0F1F060
-10          : s370fzdt5.1 F0F1F0F060
10           : s370fzdt5.0 F0F0F1F04E
10           : s370fzdt5.1 F0F1F0F04E
```

S370FZDUw.d

This format converts numeric data into z/OS zoned decimal format with no sign byte.

If present, the number of digits (d) is a scaling power of 10.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
w=5;
do n=-10,10;
  do d=0,1;
    put n @;
    s1 = putn(n,'s370fzdu',w,d);
    s2 = putc(s1,'$hex',w*2);
    put @20 ': s370fzdu' w +(-1) '.' d s2;
  end;
end;
run;
```

Which produces the following output in the log:

```
-10          : s370fzdu5.0 F0F0F0F1F0
-10          : s370fzdu5.1 F0F0F1F0F0
 10          : s370fzdu5.0 F0F0F0F1F0
 10          : s370fzdu5.1 F0F0F1F0F0
```

SSNw.

This format converts its numeric input to US social security number format.

	min	max	default
Variable width	11	11	11

Example

```
data _null_;
d=10;
d1=123456789;
put d ssn. "*";
put d1 ssn. "*";
run;
```

Which produces the following output in the log:

```
000-00-0010*  
123-45-6789*
```

WORDFw.

The format converts its numeric input into English words, assuming the output width is specified as sufficiently large. Fractions are shown as a number of one-hundredths.

	min	max	default
Variable width	5	32767	10

Example

```
data _null_;  
d=10.2;  
d1=12345.3;  
put d wordf. " ";  
put d1 wordf. " ";  
put d wordf40. " ";  
put d1 wordf80. " ";  
run;
```

Which produces the following output in the log:

```
ten and 2**  
twelve th**  
ten and 20/100 *  
twelve thousand three hundred forty-five and 30/100 *
```

WORDS_w.

This format converts its numeric input to English words, assuming the output width is specified as sufficiently large.

	min	max	default
Variable width	5	32767	10

Example

```
data _null_;  
d=10;  
d1=12345;  
put d words. "*";  
put d1 words. "*";  
put d words40. "*";  
put d1 words80. "*";  
run;
```

Which produces the following output in the log:

```
ten          *  
twelve th**  
ten          *  
twelve thousand three hundred forty-five          *
```

Zw.d

This format pads its numeric input with leading zeroes rather than blanks, if enough output space is available.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
d=10;  
d1=12345;  
put d z5. "*";  
put d1 z5. "*";  
put d z20. "*";  
put d1 z20. "*";  
run;
```

Which produces the following output in the log:

```
00010*  
12345*  
0000000000000000000010*  
0000000000000000012345*
```


ZDw.d

This format converts a numeric input into a zoned decimal format with one digit (or character) per byte. The result is ultimately platform-dependent.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
do n = -10, 1, 1234567.95, ., .A, 1E60;
  put n @;
  do w = 3, 4, 7;
    do d = 0, 4;
      s1 = putn(n, 'zd', w, d);
      s2 = putc(s1, '$hex', w*2);
      put @20 ': zd' w +(-1) '.' d s2;
    end;
  end;
end;
run;
```

Which produces the following output in the log

```
-10          : zd3.0 30317D
              : zd3.4 393952
              : zd4.0 3030317D
              : zd4.4 39393952
              : zd7.0 3030303030317D
              : zd7.4 3031303030307D
1            : zd3.0 303041
              : zd3.4 393949
              : zd4.0 30303041
              : zd4.4 39393949
              : zd7.0 30303030303041
              : zd7.4 3030313030307B
1234567.95  : zd3.0 393949
              : zd3.4 393949
              : zd4.0 39393949
              : zd4.4 39393949
              : zd7.0 31323334353648
              : zd7.4 39393939393949
.            : zd3.0 30307D
              : zd3.4 30307D
              : zd4.0 3030307D
              : zd4.4 3030307D
              : zd7.0 3030303030307D
              : zd7.4 3030303030307D
A            : zd3.0 30307D
              : zd3.4 30307D
              : zd4.0 3030307D
              : zd4.4 3030307D
              : zd7.0 3030303030307D
              : zd7.4 3030303030307D
```

```
1E60      : zd3.0 393949
           : zd3.4 393949
           : zd4.0 39393949
           : zd4.4 39393949
           : zd7.0 39393939393949
           : zd7.4 39393939393949
```

Numeric date formats

Formats that convert a numeric date into a formatted character representation.

DATEw.

This format converts a numeric date into a DDMMM, DDMMYY, DDMMYYYY or DD-MMM-YYYY form, depending upon the specified output width.

	min	max	default
Variable width	5	11	7

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with white space.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d date.;
  put explanation "5  " d date5.;
  put explanation "7  " d date7.;
  put explanation "9  " d date9.;
  put explanation "11 " d date11.;
run;
```

Which produces the following output in the log:

```
Output length: default 29SEP14
Output length: 5  29SEP
Output length: 7  29SEP14
Output length: 9  29SEP2014
Output length: 11 29-SEP-2014
```

DATEAMPM $w.d$

This format converts a numeric datetime into a visual form that depends upon the specified output length.

When the format is set to more than 19 characters, the output is padded at the front of the string with whitespace. If sufficient room is available, an AM or PM suffix is added to the result.

	min	max	default
Variable width	7	40	19

Example

```
data _null_;  
  d=1727610480;  
  explanation = "Output length: ";  
  put explanation "default " d dateampm.;  
  put explanation "7 " d dateampm7.;  
  put explanation "9 " d dateampm9.;  
  put explanation "10 " d dateampm10.;  
  put explanation "13 " d dateampm13.;  
  put explanation "16 " d dateampm16.;  
  put explanation "19 " d dateampm19.;  
  put explanation "40 " d dateampm40.;  
run;
```

Which produces the following output in the log:

```
Output length: default 29SEP14:11:48:00 AM  
Output length: 7 29SEP14  
Output length: 9 29SEP2014  
Output length: 10 29SEP14:11  
Output length: 13 29SEP14:11 AM  
Output length: 16 29SEP14:11:48 AM  
Output length: 19 29SEP14:11:48:00 AM  
Output length: 40 29SEP2014:11:48:00 AM
```

DATETIME $w.d$

This format converts a numeric datetime into a visual form that depends upon the specified output length. Time is expressed in the 24-hour clock format.

	min	max	default
Variable width	7	40	16

Example

```
data _null_;
  d=1727610480;
  explanation = "Output length: ";
  put explanation "default " d datetime.;
  put explanation "7  " d datetime7.;
  put explanation "9  " d datetime9.;
  put explanation "10 " d datetime10.;
  put explanation "13 " d datetime13.;
  put explanation "16 " d datetime16.;
  put explanation "40 " d datetime40.;
run;
```

Which produces the following output in the log:

```
Output length: default 29SEP14:11:48:00
Output length: 7  29SEP14
Output length: 9  29SEP2014
Output length: 10 29SEP14:11
Output length: 13 29SEP14:11:48
Output length: 16 29SEP14:11:48:00
Output length: 40                               29SEP2014:11:48:00
```

DAYw.

This format converts a numeric date into a number representing the day of the month on which it occurred.

	min	max	default
Variable width	2	32	2

Example

```
data _null_;
  d=19995; /* 29 Sept 2014 */
  put d day.;
run;
```

Which produces the following output in the log:

```
29
```

DDMMYYw.

This format converts a numeric date into a DD/MM/YY representation, with variations dependent upon the required output length.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "default " d ddmmyy.;  
  put explanation "2  " d ddmmyy2.;  
  put explanation "4  " d ddmmyy4.;  
  put explanation "5  " d ddmmyy5.;  
  put explanation "6  " d ddmmyy6.;  
  put explanation "8  " d ddmmyy8.;  
  put explanation "10 " d ddmmyy10.;  
run;
```

Which produces the following output in the log:

```
Output length: default 29/09/14  
Output length: 2  29  
Output length: 4  2909  
Output length: 5  29/09  
Output length: 6  290914  
Output length: 8  29/09/14  
Output length: 10 29/09/2014
```

DDMMYYBw.

This format converts a numeric date into a DD MM YY or DD MM YYYY representation, with variations depending on the available output space.

	min	max	default
Variable width	2	10	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with white space.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d ddmmyyb.;
  put explanation "2  " d ddmmyyb2.;
  put explanation "4  " d ddmmyyb4.;
  put explanation "5  " d ddmmyyb5.;
  put explanation "6  " d ddmmyyb6.;
  put explanation "8  " d ddmmyyb8.;
  put explanation "10 " d ddmmyyb10.;
run;
```

Which produces the following output in the log:

```
Output length: default 29 09 14
Output length: 2  29
Output length: 4  2909
Output length: 5  29 09
Output length: 6  290914
Output length: 8  29 09 14
Output length: 10 29 09 2014
```

DDMMYYDw.

This format converts a numeric date into a DD-MM-YY or DD-MM-YYYY representation, with variations according to the available space.

	min	max	default
Variable width	2	10	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with whitespace.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d ddmmyyd.;
  put explanation "2  " d ddmmyyd2.;
  put explanation "4  " d ddmmyyd4.;
  put explanation "5  " d ddmmyyd5.;
  put explanation "6  " d ddmmyyd6.;
  put explanation "8  " d ddmmyyd8.;
  put explanation "10 " d ddmmyyd10.;
run;
```

Which produces the following output in the log:

```
Output length: default 29-09-14
Output length: 2 29
Output length: 4 2909
Output length: 5 29-09
Output length: 6 290914
Output length: 8 29-09-14.
Output length: 10 29-09-2014
```

DDMMYYCw.

This format converts a numeric date into a DD:MM:YY or DD:MM:YYYY representation, with variations depending on the required output length.

	min	max	default
Variable width	2	10	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with whitespace.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d ddmmmyyc.;
  put explanation "2 " d ddmmmyyc2.;
  put explanation "4 " d ddmmmyyc4.;
  put explanation "5 " d ddmmmyyc5.;
  put explanation "6 " d ddmmmyyc6.;
  put explanation "8 " d ddmmmyyc8.;
  put explanation "10 " d ddmmmyyc10.;
run;
```

Which produces the following output in the log:

```
Output length: default 29:09:14
Output length: 2 29
Output length: 4 2909
Output length: 5 29:09
Output length: 6 290914
Output length: 8 29:09:14
Output length: 10 29:09:2014
```

DDMMYYN_w.

This format converts a numeric date into a DDMMYY or DDMMYYYY representation, with variations depending on the specified output length.

	min	max	default
Variable width	2	8	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with whitespace.

```
data _null_;
d=19995;
explanation = "Output length: ";
put explanation "default " d ddmmyyyn.;
put explanation "2 " d ddmmyyyn2.;
put explanation "4 " d ddmmyyyn4.;
put explanation "6 " d ddmmyyyn6.;
put explanation "8 " d ddmmyyyn8.;
run;
```

Which produces the following output in the log:

```
Output length: default 29092014
Output length: 2 29
Output length: 4 2909
Output length: 6 290914
Output length: 8 29092014
```

DDMMYYP_w.

This format converts a numeric date into a DD.MM.YY representation, with variations depending on the available output space.

	min	max	default
Variable width	2	10	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with whitespace.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d ddmmmyyp. "*";
  put explanation "2  " d ddmmmyyp2. "*";
  put explanation "4  " d ddmmmyyp4. "*";
  put explanation "5  " d ddmmmyyp5. "*";
  put explanation "6  " d ddmmmyyp6. "*";
  put explanation "8  " d ddmmmyyp8. "*";
run;
```

Which produces the following output in the log:

```
Output length: default 29092014
Output length: 2  29
Output length: 4  2909
Output length: 6  290914
Output length: 8  29092014
```

DDMMYYSw.

This format converts a numeric date into a DD/MM/YY or DD/MM/YYYY representation, with variations depending on the specified output length.

	min	max	default
Variable width	2	10	8

Example

This example shows the points where the format width selected changes the presentation of a date format. Using other format output widths pads the start of the date presentation with white space.

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "default " d ddmmmyys.;
  put explanation "2  " d ddmmmyys2.;
  put explanation "4  " d ddmmmyys4.;
  put explanation "5  " d ddmmmyys5.;
  put explanation "6  " d ddmmmyys6.;
  put explanation "8  " d ddmmmyys8.;
  put explanation "10 " d ddmmmyys10.;
run;
```

Which produces the following output in the log:

```
Output length: default 29/09/14
Output length: 2 29
Output length: 4 2909
Output length: 5 29/09
Output length: 6 290914
Output length: 8 29/09/14
Output length: 10 29/09/2014
```

DOWNNAMEw.

This format converts a numeric date into a spelled-out day of the week. It always produces the result in English – 'Monday', 'Tuesday' and so on. When the complete result is longer than the available space, it is simply truncated.

	min	max	default
Variable width	1	32	9

Example

```
data _null_;
  d=19995;
  put d downname. ;
  put d downname2. ;
run;
```

Which produces the following output in the log:

```
Monday
Mo
```

DTDATEw.

This format converts a numeric datetime into a DDMMYY or DDMMYYYY representation of the date component of its input value.

	min	max	default
Variable width	5	9	7

Example

```
data _null_;
  dt=1727610480;
  put dt dtdate9.;
run;
```

Which produces the following output in the log:

```
29SEP2014
```

DTMONYYw.

This format converts a numeric datetime into a MMMYY or MMMYYYY representation of the date part.

	min	max	default
Variable width	5	7	5

Example

```
data _null_;
  d=1727610480;
  put d dtmonyy7. ;
  put d dtmonyy. ;
run;
```

Which produces the following output in the log:

```
SEP2014
SEP14
```

DTWKDATXw.

This format converts a numeric datetime into day-of-week, day-number month-name YYYY or day-of-week, day-number MMM YYYY.

	min	max	default
Variable width	3	37	29

Example

```
data _null_;
  d=1727610480;
  put d dtwkdatx25.;
  put d dtwkdatx.;
run;
```

Which produces the following output in the log:

```
Monday, 29 Sep 2014
Monday, 29 September 2014
```

DTYEARw.

This format converts the date part of a numeric datetime into a YYYY representation. If the output space is less than 4, the last two digits of the year are displayed.

	min	max	default
Variable width	2	4	4

Example

```
data _null_;
  d=1727610480;
  put d dtyear2.;
  put d dtyear3.;
  put d dtyear4.;
run;
```

Which produces the following output in the log

```
14
 14
2014
```

DTYYQCw.

This format converts the date part of a numeric datetime into a YY:Q or YYYY:Q representation where Q is the quarter in which the date component of the input value falls.

	min	max	default
Variable width	4	6	4

Example

```
data _null_;
  d=1727610480;
  put d dtYYQC6.;
  put d dtYYQC4.;
run;
```

Which produces the following output in the log:

```
2014:3
14:3
```

HHMM*w.d*

This format converts a numeric time into an HH:MM representation.

	min	max	default
Variable width	2	20	5

Example

```
data _null_;
  d=0;
  d1=3601;
  d2=3600*15;
  put d hhmm. "*";
  put d1 hhmm20. "*";
  put d2 hhmm. "*";
run;
```

Which produces the following output in the log:

```
0:00*
          1:00*
15:00*
```

HOURL*w.d*

This format converts a numeric time into a number representing the hour of the day.

	min	max	default
Variable width	2	20	2

Example

```
data _null_;
  d=0;
  d1=3601;
  d2=3600*15;
  put d hour. "*";
  put d1 hour20. "*";
  put d2 hour. "*";
run;
```

Which produces the following output in the log:

```
0*
          1*
15*
```

JULDAY_w.

This format converts a numeric date into a number representing the Julian day of the year.

	min	max	default
Variable width	3	32	3

Example

```
data _null_;  
d=19995;  
put d julday. ;  
run;
```

Which produces the following output in the log:

```
272
```

JULIAN_w.

This format converts a numeric date into a YYDDD or YYYYDDD Julian day format.

	min	max	default
Variable width	5	7	5

Example

```
data _null_;  
d=19995;  
put d julian7.;  
put d julian5.;  
run;
```

Which produces the following output in the log:

```
2014272  
14272
```

JULDATE_w.

Converts a numeric date into YYDDD or YYYYDDD Julian day format.

This format is an alias of JULIAN.

	min	max	default
Variable width	5	7	5

Example

```
data _null_;  
d=19995;  
put d julian7.;  
run;
```

Which produces the following output in the log:

```
2014272
```

MDYAMP Mw.d

This format converts a numeric datetime value into a MM/DD/YYYY form, or MM/DD/YY if the available output space is too short.

	min	max	default
Variable width	8	40	19

Example

```
data _null_;  
n=1727610480;  
explanation = "Output length: ";  
put explanation "8 " n mdyampm8.;  
put explanation "10 " n mdyampm10.;  
put explanation "19 " n mdyampm19.;  
put explanation "30 " n mdyampm30.;  
put explanation "40 " n mdyampm40.;  
run;
```

Which produces the following output in the log:

```
Output length: 8 09/29/14  
Output length: 10 09/29/2014  
Output length: 19 09/29/2014 11:48 AM  
Output length: 30 09/29/2014 11:48 AM  
Output length: 40 09/29/2014 11:48 AM
```

MINGUOw.

This format converts a numeric date into a Taiwanese date format (which counts years from a start in 1911), with visual variations depending on the specified output width.

	min	max	default
Variable width	6	10	8

Example

```
data _null_;  
d=19995;  
put d minguo.;  
put d minguo7.;  
put d minguo10.;  
run;
```

Which produces the following output in the log:

```
1030929  
1030929  
0103/09/29
```

MMDDYYw.

This format converts a numeric date into a MM/DD/YY or MM/DD/YYYY format with variations according to the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
d=19995;  
explanation = "Output length: ";  
put explanation "2 " d mmddyy2.;  
put explanation "4 " d mmddyy4.;  
put explanation "5 " d mmddyy5.;  
put explanation "6 " d mmddyy6.;  
put explanation "8 " d mmddyy8.;  
put explanation "10 " d mmddyy10.;  
run;
```


Which produces the following output in the log:

```
Output length: 2 09
Output length: 4 0929
Output length: 5 09/29
Output length: 6 092914
Output length: 8 09/29/14
Output length: 10 09/29/2014
```

MMDDYYBw.

This format converts a numeric date into a MM DD YY or MM DD YYYY format with variations depending on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "2 " d mmddyyb2.;
  put explanation "4 " d mmddyyb4.;
  put explanation "5 " d mmddyyb5.;
  put explanation "6 " d mmddyyb6.;
  put explanation "8 " d mmddyyb8.;
  put explanation "10 " d mmddyyb10.;
run;
```

Which produces the following output in the log:

```
Output length: 2 09
Output length: 4 0929
Output length: 5 09 29
Output length: 6 092914
Output length: 8 09 29 14
Output length: 10 09 29 2014
```

MMDDYYCw.

This format converts a numeric date into a MM:DD:YY or MM:DD:YYYY form with variations depending on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "2  " d mmddyyc2.;
  put explanation "4  " d mmddyyc4.;
  put explanation "5  " d mmddyyc5.;
  put explanation "6  " d mmddyyc6.;
  put explanation "8  " d mmddyyc8.;
  put explanation "10 " d mmddyyc10.;
run;
```

Which produces the following output in the log:

```
Output length: 2  09
Output length: 4  0929
Output length: 5  09:29
Output length: 6  092914
Output length: 8  09:29:14
Output length: 10 09:29:2014
```

MMDDYYDw.

This format converts a numeric date into a MM-DD-YY or MM-DD-YYYY form with variations depending on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "2  " d mmddydd2.;
  put explanation "4  " d mmddydd4.;
  put explanation "5  " d mmddydd5.;
  put explanation "6  " d mmddydd6.;
  put explanation "8  " d mmddydd8.;
  put explanation "10 " d mmddydd10.;
run;
```

Which produces the following output in the log:

```
Output length: 2  09
Output length: 4  0929
Output length: 5  09-29
Output length: 6  092914
Output length: 8  09-29-14
Output length: 10 09-29-2014
```

MMDDYYN_w.

This format converts a numeric date into a MMDDYY or MMDDYYYY form with variations depending on the specified output width.

	min	max	default
Variable width	2	8	8

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "2 " d mmddyyn2.;  
  put explanation "4 " d mmddyyn4.;  
  put explanation "5 " d mmddyyn5.;  
  put explanation "6 " d mmddyyn6.;  
  put explanation "8 " d mmddyyn8.;  
run;
```

Which produces the following output in the log:

```
Output length: 2 09  
Output length: 4 0929  
Output length: 5 0929  
Output length: 6 092914  
Output length: 8 09292014
```

MMDDYYP_w.

This format converts a numeric date into a MM.DD.YY or MM.DD.YYYY form with variations depending on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "2 " d mmddyyp2.;  
  put explanation "4 " d mmddyyp4.;  
  put explanation "5 " d mmddyyp5.;  
  put explanation "6 " d mmddyyp6.;  
  put explanation "8 " d mmddyyp8.;  
  put explanation "10 " d mmddyyp10.;  
run;
```

Which produces the following output in the log:

```
Output length: 2 09
Output length: 4 0929
Output length: 5 09.29
Output length: 6 092914
Output length: 8 09.29.14
Output length: 10 09.29.2014
```

MMDDYYSw.

This format converts a numeric date into a MM/DD/YY or MM/DD/YYYY form with variations depending on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
  d=19995;
  explanation = "Output length: ";
  put explanation "2 " d mmddyys2.;
  put explanation "4 " d mmddyys4.;
  put explanation "5 " d mmddyys5.;
  put explanation "6 " d mmddyys6.;
  put explanation "8 " d mmddyys8.;
  put explanation "10 " d mmddyys10.;
run;
```

Which produces the following output in the log:

```
Output length: 2 09
Output length: 4 0929
Output length: 5 09/29
Output length: 6 092914
Output length: 8 09/29/14
Output length: 10 09/29/2014
```

MMSSw.d

This format converts a numeric time into a MM:SS form.

	min	max	default
Variable width	2	20	5

Example

```
data _null_;
  t=0;
  t1= 3601;
  put t mmss.;
  put t mmss19.;
  put t mmss19.5;
  put t1 mmss.;
run;
```

Which produces the following output in the log

```
0:00
          0:00
0:00.00000
60:01
```

MMYYw.

This format converts a numeric date into a nnMyyyy form where nn is the month number.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;
  d=19995;
  put d mmyy8.;
  put d mmyy7.;
  put d mmyy5.;
run;
```

Which produces the following output in the log:

```
09M2014
09M2014
09M14
```

MMYYCw.

This format converts a numeric date into a MM:YY or MM:YYYY form.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "5  " d mmyyc5.;  
  put explanation "7  " d mmyyc7.;  
  put explanation "8  " d mmyyc8.;  
run;
```

Which produces the following output in the log:

```
Output length: 5  09:14  
Output length: 7  09:2014  
Output length: 8  09:2014
```

MMYYDw.

This format converts a numeric date into a MM-YY or MM-YYYY form.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "5  " d mmyyd5.;  
  put explanation "7  " d mmyyd7.;  
  put explanation "8  " d mmyyd8.;  
run;
```

Which produces the following output in the log:

```
Output length: 5  09-14  
Output length: 7  09-2014  
Output length: 8  09-2014
```

MMYYNw.

This format converts a numeric date into a MMYN or MMYYYY form.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
d=19995;  
put d mmyyn8.;  
put d mmyyn7.;  
put d mmyyn6.;  
put d mmyyn5.;  
put d mmyyn4.;  
run;
```

Which produces the following output in the log:

```
092014  
092014  
092014  
0914  
0914
```

MMYYPw.

This format converts a numeric date into a MM.YY or MM.YYYY form.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
d=19995;  
explanation = "Output length: ";  
put explanation "5 " d mmyyp5.;  
put explanation "7 " d mmyyp7.;  
run;
```

Which produces the following output in the log:

```
Output length: 5 09.14  
Output length: 7 09.2014
```

MMYYSw.

This format converts a numeric date into a MM/YY or MM/YYYY form.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  d=19995;  
  explanation = "Output length: ";  
  put explanation "5  " d mmyys5.;  
  put explanation "7  " d mmyys7.;  
run;
```

Which produces the following output in the log:

```
Output length: 5  09/14  
Output length: 7  09/2014
```

MONNAMEw.

This format converts a numeric date into a spelled-out month - sufficient output space must be available if truncation is to be avoided. The output is not locale-sensitive – it always generates 'January', 'February' and so on.

	min	max	default
Variable width	1	32	9

Example

```
data _null_;  
  d=19995;  
  put "Default:          " d monname.;  
  put "Output length 9: " d monname9.;  
run;
```

Which produces the following output in the log:

```
Default:          September  
Output length 9: September
```

MONTHw.

This format extracts the month number from a numeric date.

	min	max	default
Variable width	1	32	2

Example

```
data _null_;
d=19995;
put d month.;
run;
```

Which produces the following output in the log:

```
9
```

MONYYw.

This format converts a numeric date into a MMMYY or MMMYYYYY format.

	min	max	default
Variable width	5	7	5

Example

```
data _null_;
d=19995;
explanation = "Output length: ";
put explanation "5 " d monyy5.;
put explanation "7 " d monyy7.;
run;
```

Which produces the following output in the log:

```
Output length: 5 SEP14
Output length: 7 SEP2014
```

NENGOW.

This format converts a numeric date into a Japanese date format including an initial era signifier. Variations in presentation depend on the specified output width.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
  d=19995;
  put "Default:          " d nengo.;
  put "Output length 5:  " d nengo5.;
  put "Output length 10: " d nengo10.;
run;
```

Which produces the following output in the log:

```
Default:          H.260929
Output length 5:  H26
Output length 10: H.26/09/29
```

PDJULGw.

This format converts a numeric date into a packed decimal representation of a Julian date.

	min	max	default
Variable width	3	16	4

Example

```
data _null_;
  length s1 $ 8 s2 $ 16;
  do n = 19995;
    put n @;
    do w = 3,7,15;
      do d = 0;
        s1 = putn(n,'pdjulg',w,d);
        s2 = putc(s1,'$hex',w*2);
        put @20 ': pdjulg' w +(-1) '.' d s2;
      end;
    end;
  end;
run;
```

Which produces the following output in the log:

```
19995          : pdjulg3.0 99992F
                : pdjulg7.0 0000002014272F
                : pdjulg15.0 000000002014272F
```

PDJULIw.

This format converts a numeric date into a packed decimal representation of a Julian date. When space permits, it also prepends a numeric representation of the century to the output, rebased to 1900.

	min	max	default
Variable width	3	16	4

Example

```
data _null_;
length s1 $ 8 s2 $ 16;
do n = 19995, 50000;
  put n @;
  do w = 3,7,15;
    do d = 0;
      s1 = putn(n,'pdjuli',w,d);
      s2 = putc(s1,'$hex',w*2);
      put @20 ': pdjuli' w +(-1) '.' d s2;
    end;
  end;
end;
run;
```

Which produces the following output in the log:

```
19995          : pdjuli3.0 99992F
                : pdjuli7.0 0000000114272F
                : pdjuli15.0 000000000114272F
50000          : pdjuli3.0 99997F
                : pdjuli7.0 0000000196327F
                : pdjuli15.0 000000000196327F
```

QTRw.

This format converts a numeric date value into an integer representing the quarter in which the date occurs.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;
  n=19995;
  put n qtr.;
run;
```

Which produces the following output in the log:

```
3
```

QTRRw.

This format converts a numeric date value into a Roman numeral representing the quarter in which the date occurs.

	min	max	default
Variable width	3	32	3

Example

```
data _null_;  
  n=19995;  
  put n qtrr.;  
run;
```

Which produces the following output in the log:

```
III
```

TIMEw.d

This format converts a numeric time value into a representation of a time interval in HH:MM:SS format, with small variations according to the available space.

	min	max	default
Variable width	2	20	8

Example

```
data _null_;  
  d=19995;  
  put "Default:           " d time.;  
  put "Output length 2:  " d time2.;  
  put "Output length 5:  " d time5.;  
run;
```

Which produces the following output in the log:

```
Default:           5:33:15  
Output length 2:  5  
Output length 5:  5:33
```

TIMEAMPM $w.d$

This format converts a numeric time value into a representation of the time of day in HH:MM:SS AM or HH:MM:SS PM formats. Small variations adjust for the available space.

	min	max	default
Variable width	2	20	8

Example

```
data _null_;
  n=3000;
  put "Default:           " n timeampm.;
  put "Output length 5:  " n timeampm5.;
  put "Output length 8:  " n timeampm8.;
run;
```

Which produces the following output in the log:

```
Default:           12:50:00 AM
Output length 5:  12 AM
Output length 8:  12:50 AM
```

TOD $w.d$

This format converts a numeric time value into a representation of the time of day in HH:MM:SS format using the 24-hour clock.

	min	max	default
Variable width	2	20	8

Example

```
data _null_;
  n=3000;
  put "Default:           " n tod.;
  put "Output length 2:  " n tod2.;
  put "Output length 4:  " n tod4.;
  put "Output length 5:  " n tod5.;
  put "Output length 7:  " n tod7.;
  put "Output length 8:  " n tod8.;
run;
```

Which produces the following output in the log:

```
Default:          00:50:00
Output length 2:  00
Output length 4:  0:50
Output length 5:  00:50
Output length 7:  0:50:00
Output length 8:  00:50:00
```

WEEKUw.

Converts a numeric date value into a yyyy-Wxx-nn form, where xx is the number of the week in the year starting at 00, and nn is the number of the day within the week, starting with 01 on Sunday.

	min	max	default
Variable width	3	200	11

Example

```
data _null_;
n=19995;
put n weeku.;
run;
```

Which produces the following output in the log:

```
2014-W39-02
```

WEEKVw.

Converts a numeric date value into an ISO week date yyyy-Wxx-nn format, where xx is the number of the week in the year starting at 01, and nn is the number of the day within the week, starting with 01 on Monday.

	min	max	default
Variable width	3	200	11

Example

```
data _null_;
n = 19995;
put n weekv.;
run
```

Which produces the following output in the log:

```
2014-W40-01
```

WEEKWw.

Converts a numeric date value into a yyyy-Wxx-nn format, where xx is the number of the week in the year starting at 00, and nn is the number of the day within the week, starting with 01 on Monday.

	min	max	default
Variable width	3	200	11

Example

```
data _null_;
n=19995;
put n weekw.;
run;
```

Which produces the following output in the log:

```
2014-W39-01
```

WEEKDATEw.

Converts a numeric date into a spelled-out representation such as: Friday, January 1, 1960.

Variations will occur in relation to the available output space. It differs only slightly from the WEEKDATE format, which displays typical output as: Friday, 1 January, 1960.

	min	max	default
Variable width	3	37	29

Example

```
data _null_;
n=19995;
put n weekdate3.;
put n weekdate13.;
put n weekdate14.;
put n weekdate21.;
put n weekdate22.;
put n weekdate29.;
run;
```

Which produces the following output in the log:

```
Mon
    Monday
    Monday
    Mon, Sep 29, 2014
    Mon, Sep 29, 2014
    Monday, September 29, 2014
```

WEEKDATX_w.

Converts a numeric date into a spelled-out representation such as: Friday, 1 January, 1960.

The format differs only slightly from the `WEEKDATE` format, which displays typical output as: Friday, January 1, 1960.

	min	max	default
Variable width	3	37	29

Example

```
data _null_;
  n=19995;
  put "Default output length: " n weekdatx.;
  put "Output length 15:      " n weekdatx15.;
  put "Output length 17:      " n weekdatx17.;
  put "Output length 23:      " n weekdatx23.;
run;
```

Which produces the following output in the log:

```
Default output length:    Monday, 29 September 2014
Output length 15:        Mon, 29 Sep 14
Output length 17:        Mon, 29 Sep 2014
Output length 23:        Monday, 29 Sep 2014
```

WEEKDAY_w.

Converts a numeric date into a number representing the day of the week starting on Sunday.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;
n=19995;
put n weekday.;
run;
```

Which produces the following output in the log:

```
2
```

WORDDATE*w.*

Converts a numeric date into a spelled-out representation such as: January 1, 1960, with presentational variations according to the available output space.

The format differs only slightly from the WORDDATX format, in which the output takes the form: 29 September 2014.

	min	max	default
Variable width	3	32	18

Example

```
data _null_;
n=19995;
put n worddate3.;
put n worddate13.;
put n worddate14.;
put n worddate21.;
put n worddate22.;
put n worddate29.;
run;
```

Which produces the following output in the log:

```
Sep
Sep 29, 2014
Sep 29, 2014
September 29, 2014
September 29, 2014
September 29, 2014
```

WORDDATXw.

Converts a numeric date into a spelled-out representation such as: 1 January 1960, with presentational variations according to the available output space.

The format differs only slightly from the `WORDDATE` format, in which the output takes the form: September 29, 2014. This format is not locale-sensitive and the output will always be in English.

	min	max	default
Variable width	3	32	18

Example

```
data _null_;  
n=19995;  
put n worddatx3.;  
put n worddatx13.;  
put n worddatx14.;  
put n worddatx21.;  
put n worddatx22.;  
put n worddatx29.;  
run;
```

Which produces the following output in the log:

```
Sep  
29 Sep 2014  
29 Sep 2014  
29 September 2014  
29 September 2014  
29 September 2014
```

YEARw.

Transforms a numeric date into a representation of the year.

	min	max	default
Variable width	2	32	4

Example

```
data _null_;  
n=19995;  
put n year.;  
run;
```

Which produces the following output in the log:

```
2014
```

YYMMw.

Transforms a numeric date into a yyyyMxx form, where xx is the month number.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
n=19995;  
put n yymm5.;  
put n yymm.;  
run;
```

Which produces the following output in the log:

```
14M09  
2014M09
```

YYMMCw.

Transforms a numeric date into a colon-separated YYYY:MM form, where MM is the month number.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
n=19995;  
put n yymmc5.;  
put n yymmc6.;  
put n yymmc7.;  
put n yymmc8.;  
put n yymmc9.;  
put n yymmc10.;  
put n yymmc11.;  
put n yymmc12.;  
run;
```

Which produces the following output in the log:

```
14:09
 14:09
2014:09
 2014:09
 2014:09
 2014:09
 2014:09
 2014:09
 2014:09
```

YYMMDw.

Converts a numeric date into a dash-separated YYYY-MM form, where MM is the month number.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;
n=19995;
put n yymmd5.;
put n yymmd6.;
put n yymmd7.;
put n yymmd8.;
put n yymmd9.;
put n yymmd10.;
put n yymmd11.;
put n yymmd12.;
run;
```

Which produces the following output in the log:

```
14-09
 14-09
2014-09
 2014-09
 2014-09
 2014-09
 2014-09
 2014-09
 2014-09
```

YYMMNW.

This format transforms a numeric date into a YYYYMM form.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
n=19995;  
put n yymmn4.;  
put n yymmn5.;  
put n yymmn6.;  
put n yymmn7.;  
put n yymmn8.;  
put n yymmn9.;  
put n yymmn10.;  
run;
```

Which produces the following output in the log:

```
1409  
1409  
201409  
201409  
201409  
201409  
201409  
201409
```

YYMMPw.

This format transforms a numeric date into a period-separated YYYY.MM form where MM is the month number within the year.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
n=19995;  
put n yymmp5.;  
put n yymmp7.;  
run;
```

Which produces the following output in the log:

```
14.09  
2014.09
```

YYMMSw.

This format transforms a numeric date into a slash-separated YYYY/MM form where MM is the month number within the year.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  n=19995;  
  put n yymms5.;  
  put n yymms7.;  
run;
```

Which produces the following output in the log:

```
14/09  
2014/09
```

YYMMDDw.

This format transforms a numeric date into a YY-MM-DD or YYYY-MM-DD form, depending on the available space.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
  n=19995;  
  put n yymmdd2.;  
  put n yymmdd4.;  
  put n yymmdd5.;  
  put n yymmdd6.;  
  put n yymmdd8.;  
  put n yymmdd10.;  
run;
```

Which produces the following output in the log:

```
14  
1409  
14-09  
140929  
14-09-29  
2014-09-29
```

YYMMDDbw.

Transforms a numeric date into a space-separated YY MM DD or YYYY MM DD form, depending on the available space

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
n=19995;  
put n yymddb2.;  
put n yymddb4.;  
put n yymddb5.;  
put n yymddb6.;  
put n yymddb8.;  
put n yymddb10.;  
run;
```

Which produces the following output in the log:

```
14  
1409  
14 09  
140929  
14 09 29  
2014 09 29
```

YYMMDDCw.

Transforms a numeric date into a colon-separated YY:MM:DD or YYYY:MM:DD form, depending on the available space.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
n=19995;  
put n yymddc2.;  
put n yymddc4.;  
put n yymddc5.;  
put n yymddc6.;  
put n yymddc8.;  
put n yymddc10.;  
run;
```

Which produces the following output in the log:

```
14
1409
14:09
140929
14:09:29
2014:09:29
```

YYMMDDDW.

Transforms a numeric date into a dash-separated YY-MM-DD or YYYY-MM-DD form, depending on the available space.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;
n=19995;
put n yymmddd2.;
put n yymmddd4.;
put n yymmddd5.;
put n yymmddd6.;
put n yymmddd8.;
put n yymmddd10.;
run;
```

Which produces the following output in the log:

```
14
1409
14-09
140929
14-09-29
2014-09-29
```

YYMMDDNW.

Transforms a numeric date into a YYMMDD or YYYYMMDD form, depending on the available space.

	min	max	default
Variable width	2	8	8

Example

```
data _null_;  
  n=19995;  
  put n yymddn2.;  
  put n yymddn4.;  
  put n yymddn6.;  
  put n yymddn8.;  
run;
```

Which produces the following output in the log:

```
14  
1409  
140929  
20140929
```

YYMMDDP_w.

Transforms a numeric date into a period-separated YY.MM.DD or YYYY.MM.DD form, depending on the available space.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
  n=19995;  
  put n yymddp2.;  
  put n yymddp4.;  
  put n yymddp5.;  
  put n yymddp6.;  
  put n yymddp8.;  
  put n yymddp10.;  
run;
```

Which produces the following output in the log:

```
14  
1409  
14.09  
140929  
14.09.29  
2014.09.29
```

YYMMDDSw.

Transforms a numeric date into a slash-separated YY/MM/DD or YYYY/MM/DD form, depending on the available space.

	min	max	default
Variable width	2	10	8

Example

```
data _null_;  
  n=19995;  
  put n yymdds2.;  
  put n yymdds4.;  
  put n yymdds5.;  
  put n yymdds6.;  
  put n yymdds8.;  
  put n yymdds10.;  
run;
```

Which produces the following output in the log:

```
14  
1409  
14/09  
140929  
14/09/29  
2014/09/29
```

YYMONw.

Transforms a numeric date into a YYYYMMM form.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  n=19995;  
  put n yymon5.;  
  put n yymon7.;  
run;
```

Which produces the following output in the log:

```
14SEP  
2014SEP
```

YYQw.

Transforms a numeric date into a YYYYQxx form, where xx is the quarter in which the original date falls.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
  n=19995;  
  put n yyq4.;  
  put n yyq6.;  
run;
```

Which produces the following output in the log:

```
14Q3  
2014Q3
```

YYQCw.

Transforms a numeric date into a YYYY:Q form, where Q is the number of the quarter in which the original date falls.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
  n=19995;  
  put n yyqc4.;  
  put n yyqc6.;  
run;
```

Which produces the following output in the log:

```
14:3  
2014:3
```

YYQDw.

Transforms a numeric date into a YYYY-Q form, where Q is the number of the quarter in which the original date falls.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
  n=19995;  
  put n yyqd4.;  
  put n yyqd6.;  
run;
```

Which produces the following output in the log:

```
14-3  
2014-3
```

YYQNw.

Converts a numeric date into a YYYYQ form, where Q is the number of the quarter in which the original date falls.

	min	max	default
Variable width	3	32	5

Example

```
data _null_;  
  n=19995;  
  put n yyqn.;  
run;
```

Which produces the following output in the log:

```
20143
```

YYQPw.

Transforms a numeric date into a YYYY.Q form, where Q is the number of the quarter in which the date falls.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
  n=19995;  
  put n yyqp.;  
run;
```

Which produces the following output in the log:

```
2014.3
```

YYQRw.

Converts a numeric date into a YYYYQr form, where r is the Roman numeral representation of the quarter in which the original date falls.

	min	max	default
Variable width	6	32	8

Example

```
data _null_;  
  n=19995;  
  put n yyqr.;  
run;
```

Which produces the following output in the log:

```
2014QIII
```

YYQSw.

Transforms a numeric date into a YYYY/Q form, where Q is the number of the quarter in which the original date falls.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;
  n=19995;
  put n yyqs.;
run;
```

Which produces the following output in the log:

```
2014/3
```

YYQRCw.

Transforms a numeric date into a YYYY:r form, where r is the Roman numeral representation of the number of the quarter in which the date falls.

	min	max	default
Variable width	6	32	8

Example

```
data _null_;
  n=19995;
  put n yyqrc.;
run;
```

Which produces the following output in the log:

```
2014:III
```

YYQRDw.

Converts a numeric date into a YYYY-r form, where r is the Roman numeral representation of the number of the quarter in which the date falls.

	min	max	default
Variable width	6	32	8

Example

```
data _null_;  
  n=19995;  
  put n yyqrd.;  
run;
```

Which produces the following output in the log:

```
2014-III
```

YYQRNw.

Transforms a numeric date into a YYYYr form, where r is the Roman numeral representation of the number of the quarter in which the date falls.

	min	max	default
Variable width	5	32	7

Example

```
data _null_;  
  n=19995;  
  put n yyqrn.;  
run;
```

Which produces the following output in the log:

```
2014III
```

YYQRPw.

Transforms a numeric date into a YYYY.r form, where r is the Roman numeral representation of the number of the quarter in which the date falls.

	min	max	default
Variable width	6	32	8

Example

```
data _null_;  
  n=19995;  
  put n yyqrp.;  
run;
```

Which produces the following output in the log:

```
2014.III
```

YYQRSw.

Transforms a numeric date into a YYYY/r form, where r is the Roman numeral representation of the number of the quarter in which the date falls.

	min	max	default
Variable width	6	32	8

Example

```
data _null_;  
  n=19995;  
  put n yyqrs.;  
run;
```

Which produces the following output in the log:

```
2014/III
```

YYWEEKUw.

Transforms a numeric date into a YYYYWxx or YYYY-Wxx form, where xx is the week number.

	min	max	default
Variable width	3	8	7

Example

```
data _null_;  
  n=19995;  
  put n yyweeku.;  
  put n yyweeku8.;  
run;
```

Which produces the following output in the log:

```
2014W39  
2014-W39
```

YYWEEKVw.

Converts a numeric date value into an ISO week date YYYYWxx or YYYY-Wxx form, where xx is the number of the week in the year starting at 01.

	min	max	default
Variable width	3	8	7

Example

```
data _null_;  
  n=19995;  
  put n yyweekv.;  
  put n yyweekv8.;  
run;
```

Which produces the following output in the log:

```
2014W40  
2014-W40
```

YYWEEKWw.

Converts a numeric date value into a YYYYWxx or YYYY-Wxx form, where xx is the number of the week in the year starting at 00.

	min	max	default
Variable width	3	8	7

Example

```
data _null_;
  n=19995;
  put n yyweekw.;
  put n yyweekw8.;
run;
```

Which produces the following output in the log:

```
2014W39
2014-W39
```

ISO8601 date formats

Formats that represent date-time data according to the ISO 8601 standard.

\$N8601Bw.

Taking input in an encoded form (an internal representation), this format generates datetimes, durations and intervals as ISO 8601 basic forms `PnYnMnDnTnHnMnS` and `yyyymmddThhmmss.`

	min	max	default
Variable width	1	200	55

Example

In the following example, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
  length dur $25;
  length datim $25;
  length intval $50;
  beg='02jan2012:11:30:21'dt;
  fin='08apr2014:13:32:22'dt;
  call is8601_convert('dt/dt','du',beg,fin,dur);
  call is8601_convert('dt','dt',beg,datim);
  call is8601_convert('dt/dt','intvl',beg,fin,intval);
  put dur=$n8601b50.;
  put datim=$n8601b50.;
  put intval=$n8601b50.;
run;
```

Which produces the following output in the log:

```
dur=P2Y3M6DT2H2M1S
datim=20120102T113021
intval=20120102T113021/20140408T133222
```

\$N8601BAw.

Taking input in an encoded form (i.e. an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `PyyyymmddThhmmss` and `yyyymmddThhmmss`.

	min	max	default
Variable width	1	200	55

Example

In the example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30:21'dt;
fin='08apr2014:13:32:22'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601ba50.;
put datim=$n8601ba50.;
put intval=$n8601ba50.;
run;
```

Which produces the following output in the log:

```
dur=P00020306T020201
datim=20120102T113021
intval=20120102T113021/20140408T133222
```

\$N8601Ew.

Taking input in an encoded form (an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `PnYnMnDTnHnMnS` and `yyyy-mmddThh:mm:ss`.

	min	max	default
Variable width	1	200	55

Example

In the example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30:21'dt;
fin='08apr2014:13:32:22'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);          /* Convert to a duration */
call is8601_convert('dt','dt',beg,datim);              /* Convert to a datetime */
call is8601_convert('dt/dt','intvl',beg,fin,intval);    /* Convert to an interval */
put dur=$n8601e50.;
put datim=$n8601e50.;
put intval=$n8601e50.;
run;
```

Which produces the following output in the log:

```
dur=P2Y3M6DT2H2M1S
datim=2012-01-02T11:30:21
intval=2012-01-02T11:30:21/2014-04-08T13:32:22
```

\$N8601EAw.

Taking input in an encoded form (an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `Pyyyy-mm-ddThh:mm:ss` and `yyyy-mm-ddThh:mm:ss`.

	min	max	default
Variable width	1	200	55

Example

In the example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30:21'dt;
fin='08apr2014:13:32:22'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601ea50.;
put datim=$n8601ea50.;
put intval=$n8601ea50.;
run;
```

Which produces the following output in the log:

```
dur=P0002-03-06T02:02:01
datim=2012-01-02T11:30:21
intval=2012-01-02T11:30:21/2014-04-08T13:32:22
```

\$N8601EHw.

Taking input in an encoded form (an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `Pyyyy-mm-ddThh:mm:ss` and `yyyy-mm-ddThh:mm:ss`. This is the same as the `$N8601EAW.` format, except that it uses a hyphen for missing components.

	min	max	default
Variable width	1	200	55

Example

In the following example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30'dt;
fin='08apr2014:13:32'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601eh50.;
put datim=$n8601eh50.;
put intval=$n8601eh50.;
run;
```

Which produces the following output in the log:

```
dur=P0002-03-06T02:02:-
datim=2012-01-02T11:30:00
intval=2012-01-02T11:30:00/2014-04-08T13:32:00
```

\$N8601EXw.

Taking input in an encoded form (an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `Pyyyy-mm-ddThh:mm:ss` and `yyyy-mm-ddThh:mm:ss`.

This is the same as the \$N8601EAW. format, except that it uses x for each digit of a missing component.

	min	max	default
Variable width	1	200	55

Example

In the example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30'dt;
fin='08apr2014:13:32'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601ex50.;
put datim=$n8601ex50.;
put intval=$n8601ex50.;
run;
```

Which produces the following output in the log

```
dur=P0002-03-06T02:02:x
datim=2012-01-02T11:30:00
intval=2012-01-02T11:30:00/2014-04-08T13:32:00
```

\$N8601Hw.

Taking input in an encoded form (i.e. an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `Pyyyy-mm-ddThh:mm:ss` and `yyyy-mm-ddThh:mm:ss`.

This format is the same as the \$N8601EAW. format, except that it suppresses omitted components in durations and uses a hyphen for omitted components in datetime values.

	min	max	default
Variable width	1	200	55

Example

In the following example below, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30'dt;
fin='08apr2014:13:32'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601h50.;
put datim=$n8601h50.;
put intval=$n8601h50.;
run;
```

Which produces the following output in the log:

```
dur=P2Y3M6DT2H2M-S
datim=2012-01-02T11:30:00
intval=2012-01-02T11:30:00/2014-04-08T13:32:00
```

\$N8601Xw.

Taking input in an encoded form (i.e. an internal representation), this format generates datetimes, durations and intervals as ISO 8601 forms `Pyyyy-mm-ddThh:mm:ss` and `yyyy-mm-ddThh:mm:ss`.

This format is the same as the `$N8601Eaw.` format, except that it drops omitted components in durations and uses an x for each digit of an omitted component in datetime values.

	min	max	default
Variable width	1	200	55

Example

In the following example, the DATA step function `is8601_convert` is used to convert sample textual data to an encoded form prior to it being transformed by the format itself.

```
data _null_;
length dur $25;
length datim $25;
length intval $50;
beg='02jan2012:11:30'dt;
fin='08apr2014:13:32'dt;
call is8601_convert('dt/dt','du',beg,fin,dur);
call is8601_convert('dt','dt',beg,datim);
call is8601_convert('dt/dt','intvl',beg,fin,intval);
put dur=$n8601x50.;
put datim=$n8601x50.;
put intval=$n8601x50.;
run;
```

Which produces the following output in the log:

```
dur=P2Y3M6DT2H2MxS
datim=2012-01-02T11:30:00
intval=2012-01-02T11:30:00/2014-04-08T13:32:00
```

B8601DAw.

Converts a numeric date into a basic ISO 8601 YYYYMMDD form.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;
d = 19995;
put d b8601da10.;
run;
```

Which produces the following output in the log:

```
20140929
```


E8601DAw.

Converts a numeric date into an extended ISO 8601 YYYY-MM-DD form.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d=19995;  
put d e8601da.;  
run;
```

Which produces the following output in the log:

```
2014-09-29
```

B8601DNw.

Converts a numeric datetime into a basic ISO 8601 YYYYMMDD form.

This format extracts and writes the date from the specified numeric datetime, omitting the time component.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d = 1727610480;  
put d b8601dn.;  
run;
```

Which produces the following output in the log:

```
20140929
```

E8601DNw.

Converts a numeric datetime into an extended ISO 8601 YYYY-MM-DD form.

This format extracts and writes the date from the specified numeric datetime, omitting the time component.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d = 1727610480;  
put d e8601dn10.;  
run;
```

Which produces the following output in the log:

```
2014-09-29
```

B8601DTw.d

Converts a numeric datetime into a basic ISO 8601 YYYYMMDDTHHMMSS datetime form.

	min	max	default
Variable width	15	26	19

Example

```
data _null_;  
d = 1727610480;  
put d b8601dt.;  
run;
```

Which produces the following output in the log:

```
20140929T114800
```

E8601DTw.d

Converts a numeric datetime into an extended ISO 8601 YYYY-MM-DDTHH:MM:SS datetime form.

	min	max	default
Variable width	19	26	19

Example

```
data _null_;  
d = 1727610480;  
put d e8601dt.;  
run;
```

Which outputs the following in the log:

```
2014-09-29T11:48:00
```

B8601DZw.

Converts a numeric datetime into a UTC ISO 8601 YYYYMMDDTHHMMSS+|=HHMM datetime and timezone form.

	min	max	default
Variable width	20	35	26

Example

```
data _null_;  
d = 1727610480;  
put d b8601dz35.;  
run;
```

Which produces the following output in the log:

```
20140929T114800+0000
```

E8601DZw.

Converts a numeric datetime into an extended UTC ISO 8601 YYYY-MM-DDTHH:MM:SS+|-HH:MM datetime and timezone form.

The SAS language does not have any concept of 'timezone', the timezone is either printed 00:00 or 'Z' for 'zulu'.

	min	max	default
Variable width	20	35	26

Example

```
data _null_;  
d = 1727610480;  
put d e8601dz35.;  
run;
```

Which produces the following output in the log:

```
2014-09-29T11:48:00+00:00
```

B8601LZw.

Converts a numeric time into the basic ISO 8601 form: HHMMSS+|-HHMM, by writing the time and a time-zone offset.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d = 3000;  
put d b8601lz.;  
run;
```

Which produces the following output in the log:

```
005000+0100
```

E8601LZw.

Converts a numeric time into the extended ISO 8601 form: HH:MM:SS+|-HH:MM, by writing the time and a time-zone offset.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d = 3000;  
put d e8601lz.;  
run;
```

Which produces the following output in the log:

```
00:50:00+01:00
```

B8601TMw.d

Converts a numeric time into the basic ISO 8601 form: HHMMSS.

	min	max	default
Variable width	6	15	8

Example

```
data _null_;
d = 3000;
put d b8601tm.;
run;
```

Which produces the following output in the log:

```
005000
```

E8601TMw.d

Converts a numeric time into the extended ISO 8601 form: HH:MM:SS.

	min	max	default
Variable width	8	15	8

Example

```
data _null_;
d = 3000;
put d e8601tm.;
run;
```

Which produces the following output in the log:

```
00:50:00
```

B8601TZw.

Converts a numeric time to UTC and writes it using the ISO 8601 basic time notation HHMMSS+|-HHMM.

	min	max	default
Variable width	7	20	14

Example

```
data _null_;
d = 3000;
put d b8601tz.;
run;
```

Which produces the following output in the log:

```
005000+0000
```

E8601TZw.

Converts a numeric time to UTC and writes it using the ISO 8601 extended time notation HH:MM:SS+|-HH:MM.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d = 3000;  
put d e8601tz.;  
run;
```

Which produces the following output in the log:

```
00:50:00+00:00
```

IS8601DAw.

Converts a numeric date into an extended ISO 8601 YYYY-MM-DD form.

This format is an alias of E8601DAw.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d = 19995;  
put d is8601da.;  
run;
```

Which produces the following output in the log.

```
2014-09-29
```

IS8601DNw.

Converts a numeric datetime into an extended ISO 8601 YYYY-MM-DD form - it extracts and writes the date from the specified numeric datetime, omitting the time component.

This format is an alias of E8601DNw.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d = 1727610480;  
put d is8601dn.;  
run;
```

Which produces the following output in the log:

```
2014-09-29
```

IS8601LZw.

Converts a numeric time into the extended ISO 8601 form: HH:MM:SS+|-HH:MM, by writing the time and a time-zone offset.

This format is an alias of E8601LZw.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d=3000;  
put d is8601lz.;  
run;
```

Which produces the following output in the log:

```
00:50:00+01:00
```

IS8601TMw.d

Converts a numeric time into the extended ISO 8601 form: HH:MM:SS.

This format is an alias of E8601TMw.d.

	min	max	default
Variable width	8	15	8

Example

```
data _null_;  
d=3000;  
put d is8601tm.;  
run;
```

Which produces the following output in the log:

```
00:50:00
```

IS8601TZw.

Converts a numeric time to UTC, writing it using the ISO 8601 extended time notation HH:MM:SS+|-HH:MM.

This format is an alias of E8601TZw.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d=3000;  
put d is8601tz.;  
run;
```

Which produces the following output in the log:

```
00:50:00+00:00
```

IS8601DTw.d

Converts a numeric datetime into an extended ISO 8601 YYYY-MM-DDTHH:MM:SS datetime form.

This format is an alias of E8601DTw.d.

	min	max	default
Variable width	19	26	19

Example

```
data _null_;  
d=1727610480;  
put d is8601dt.;  
run;
```


Which produces the following output in the log:

```
2014-09-29T11:48:00
```

IS8601DZw.

Converts a numeric datetime into an extended UTC ISO 8601 YYYY-MM-DDTHH:MM:SS+|=HH:MM datetime and timezone form.

This format is an alias of E8601DZw.

	min	max	default
Variable width	20	35	26

Example

```
data _null_;  
d=1727610480;  
put d is8601dz.;  
run;
```

Which produces the following output in the log:

```
2014-09-29T11:48:00+00:00
```

International date formats

Formats that represent date-time data in a specified language.

HDATEw.

Converts a numeric date into a Hebrew form, using the default 'Long Date' and 'Medium Date' patterns for the locale, resulting in an output form of d MMM y, with variations as imposed by the available space.

When displayed, it may appear rearranged in the output log according to whether or not the configured locale embodies left-to-right or right-to-left reading conventions. See http://demo.icu-project.org/icu-bin/locexp?d=en&_he_IL for further information about Hebrew date formats.

	min	max	default
Variable width	9	17	17

Example

```
data _null_;
d=19995;
put d hdate17.;
r=put(d, hdate17.);
put r $hex.;
run;
```

Which produces the following output in the log:

```
29 2014 תשס"ד
323920D791D7A1D7A4D798203230313420
```

HEBDATEw.

Converts a numeric date into a form consistent with the Hebrew calendar.

	min	max	default
Variable width	11	45	16

Example

```
data _null_;
d=19995;
put d hebdate20.;
put d hebdate25.;
put d hebdate34.;
put d hebdate45.;
run;
```

Which produces the following output in the log:

```
ה' א' / תשע"ה
ה' תשרי' תשע"ה
ה' תשרי' תשע"ה
שני' ה' תשרי' תשע"ה
```

xxxDFDDw.

Converts a numeric date into a form that contains the same calendar elements as DDMMYY, in a specific target language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian

- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian
- SVE for Swedish

The `EURDFDDw.` format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, `DFLANG`.

	min	max	default
Output widths for all languages/dialects except those listed below	2	10	8
English	2	10	10
Finnish	2	10	10

Example

```
data _null_;  
d=0;  
put  "AFR " d afrdfdd10.;  
put  "CAT " d catdfdd10.;  
put  "CRO " d crodfdd10.;  
put  "CSY " d csydfdd10.;  
put  "DAN " d dandfdd10.;  
put  "DES " d desdfdd10.;  
put  "DEU " d deudfdd10.;  
put  "ENG " d engdfdd10.;  
put  "ESP " d espdfdd10.;  
put  "FIN " d findfdd10.;  
put  "FRA " d fradfdd10.;  
put  "FRS " d frsdfdd10.;  
put  "HUN " d hundfdd10.;  
put  "ITA " d itadfdd10.;  
put  "MAC " d macdfdd10.;  
put  "NLD " d nlddfdd10.;  
put  "NOR " d nordfdd10.;  
put  "POL " d poldfdd10.;  
put  "PTG " d ptgdfdd10.;  
put  "RUS " d rusdfdd10.;  
put  "SLO " d slodfdd10.;  
put  "SVE " d svedfdd10.;  
run;
```

Which produces the following output in the log:

```
AFR 01.01.1960  
CAT 01/01/1960  
CRO 01.01.1960  
CSY 01/01/1960  
DAN 01.01.1960  
DES 01.01.1960  
DEU 01.01.1960  
ENG 01.01.1960  
ESP 01.01.1960  
FIN 01.01.1960  
FRA 01/01/1960  
FRS 01/01/1960  
HUN    60.1.1.  
ITA 01/01/1960  
MAC 01.01.1960  
NLD 01-01-1960  
NOR 01.01.1960  
POL 01-01-1960  
PTG 01/01/1960  
RUS 01.01.1960  
SLO 01.01.1960  
SVE 01.01.1960
```

xxxDFDEw.

Converts a numerical date into a form that contains the same calendar elements as DATE, in a specific target language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian
- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian
- SVE for Swedish

The EURDFDEw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

	min	max	default
Output widths for all languages/dialects except those listed below	5	9	7
Czech	10	14	12
Finnish	9	10	9
Hungarian	12	14	12

Example

```
data _null_;  
d=0;  
put  "AFR " d afrdfde9.;  
put  "CAT " d catdfde9.;  
put  "CRO " d crodfde9.;  
put  "CSY " d csydfde10.;  
put  "DAN " d dandfde9.;  
put  "DES " d desdfde9.;  
put  "DEU " d deudfde9.;  
put  "ENG " d engdfde9.;  
put  "ESP " d espdfde9.;  
put  "FIN " d findfde9.;  
put  "FRA " d fradfde9.;  
put  "FRS " d frsdfde9.;  
put  "HUN " d hundfde12.;  
put  "ITA " d itadfde9.;  
put  "MAC " d macdfde9.;  
put  "NLD " d nlddfde9.;  
put  "NOR " d nordfde9.;  
put  "POL " d poldfde9.;  
put  "PTG " d ptgdfde9.;  
put  "RUS " d rusdfde9.;  
put  "SLO " d slodfde9.;  
put  "SVE " d svedfde9.;  
run;
```

Which produces the following output in the log:

```
AFR 01Jan1960  
CAT 01Gen1960  
CRO 01sij1960  
CSY 01leden60  
DAN 01jan1960  
DES 01Jan1960  
DEU 01Jan1960  
ENG 01JAN1960  
ESP 01ene1960  
FIN 1.1.-60  
FRA 01jan1960  
FRS 01jan1960  
HUN 60.jan.1.  
ITA 01Gen1960  
MAC 01jan1960  
NLD 01jan1960  
NOR 01jan1960  
POL 01sty1960  
PTG 01jan1960  
/* Server encoding of WCYRILLIC required for RUSDFWKX */  
RUS 01ЯНБ1960  
SLO 01jan1960  
SVE 01jan1960
```

xxxDFDNw.

Converts a numerical date into a form that contains the same calendar elements as WEEKDAY (that is, a number representing the day of the week), in a specific target language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian
- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian
- SVE for Swedish

The EURDFDNw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
d=0;  
put "AFR " d afrdfdn9.;  
put "CAT " d catdfdn9.;  
put "CRO " d crodfdn9.;  
put "CSY " d csydfdn9.;  
put "DAN " d dandfdn9.;  
put "DES " d desdfdn9.;  
put "DEU " d deudfdn9.;  
put "ENG " d engdfdn9.;  
put "ESP " d espdfdn9.;  
put "FIN " d findfdn9.;  
put "FRA " d fradfdn9.;  
put "FRS " d frsdfdn9.;  
put "HUN " d hundfdn9.;  
put "ITA " d itadfdn9.;  
put "MAC " d macdfdn9.;  
put "NLD " d nlddfndn9.;  
put "NOR " d nordfdn9.;  
put "POL " d poldfdn9.;  
put "PTG " d ptgdfdn9.;  
put "RUS " d rusdfdn9.;  
put "SLO " d slodfdn9.;  
put "SVE " d svedfdn9.;  
run;
```

Which produces the following output in the log:

AFR	5
CAT	5
CRO	5
CSY	5
DAN	5
DES	5
DEU	5
ENG	5
ESP	5
FIN	5
FRA	5
FRS	5
HUN	5
ITA	5
MAC	5
NLD	5
NOR	5
POL	5
PTG	5
RUS	5
SLO	5
SVE	5

xxxDFDTw.

Converts a numerical date into a form that contains the same calendar elements as DATETIME, in a specific target language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian
- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian
- SVE for Swedish

The EURDFDTw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

	min	max	default
Output widths for all languages/dialects except those listed below	7	40	16
Czech	12	40	21
Hungarian	12	40	19

Example

```
data _null_;
d=0;
put "AFR " d afrdfdt20.;
put "CAT " d catdfdt20.;
put "CRO " d crodfdt20.;
put "CSY " d csydfdt20.;
put "DAN " d dandfdt20.;
put "DES " d desdfdt20.;
put "DEU " d deudfdt20.;
put "ENG " d engdfdt20.;
put "ESP " d espdfdt20.;
put "FIN " d findfdt20.;
put "FRA " d fradfdt20.;
put "FRS " d frsdfdt20.;
put "HUN " d hundfdt20.;
put "ITA " d itadfdt20.;
put "MAC " d macdfdt20.;
put "NLD " d nlddfdt20.;
put "NOR " d nordfdt20.;
put "POL " d poldfdt20.;
put "PTG " d ptgdfdt20.;
put "RUS " d rusdfdt20.;
put "SLO " d slodfdt20.;
put "SVE " d svedfdt20.;
run;
```

Which produces the following output in the log:

```
AFR    01Jan1960:00:00:00
CAT    01Gen1960:00:00:00
CRO    01sij1960:00:00:00
CSY    01leden60:00:00:00
DAN    01jan1960:00:00:00
DES    01Jan1960:00:00:00
DEU    01Jan1960:00:00:00
ENG    01JAN1960:00:00:00
ESP    01ene1960:00:00:00
FIN    1.1.1960:00:00:00
FRA    01jan1960:00:00:00
FRS    01jan1960:00:00:00
HUN    60.jan.1. 00:00:00
ITA    01Gen1960:00:00:00
MAC    01jan1960:00:00:00
NLD    01jan1960:00:00:00
NOR    01jan1960:00:00:00
POL    01sty1960:00:00:00
PTG    01jan1960:00:00:00
/* Server encoding of WCYRILLIC required for RUSDFWKX */
RUS    01ЯНВ1960:00:00:00
SLO    01jan1960:00:00:00
SVE    01jan1960:00:00:00
```

xxxDFDWNw.

Converts a numerical date into a form that contains the same calendar elements as DOWNAME (that is, the name of the day of the week), into a specific language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian
- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian
- SVE for Swedish

The EURDFDWNw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

	min width	max width	default width
Output widths for all languages/dialects except those listed below	1	32	9
Croatian	1	32	10
Czech	1	32	7
Danish	1	32	7

	min width	max width	default width
Swiss_German	1	32	10
German	1	32	10
English	1	32	11
Finnish	1	32	11
French	1	32	8
Macedonian	1	32	10
Norwegian	1	32	7
Polish	1	32	12
Russian	1	32	11
Swedish	1	32	7

Example

```
data _null_;
d=0;
put "AFR " d afrdfdwn20.;
put "CAT " d catdfdwn20.;
put "CRO " d crodfdwn20.;
put "CSY " d csydfdwn20.;
put "DAN " d dandfdwn20.;
put "DES " d desdfdwn20.;
put "DEU " d deudfdwn20.;
put "ENG " d engdfdwn20.;
put "ESP " d espdfdwn20.;
put "FIN " d findfdwn20.;
put "FRA " d fradfdwn20.;
put "FRS " d frsdfdwn20.;
put "HUN " d hundfdwn20.;
put "ITA " d itadfdwn20.;
put "MAC " d macdfdwn20.;
put "NLD " d nlddfdwn20.;
put "NOR " d nordfdwn20.;
put "POL " d poldfdwn20.;
put "PTG " d ptgdfdwn20.;
put "RUS " d rusdfdwn20.;
put "SLO " d slodfdwn20.;
put "SVE " d svedfdwn20.;
run;
```

Which produces the following output in the log:

```
AFR          Vrydag
CAT          Divendres
CRO          petak
CSY          ptek
DAN          fredag
DES          Freitag
DEU          Freitag
ENG          Friday
ESP          viernes
FIN          Perjantai
```

```
FRA      Vendredi
FRS      Vendredi
HUN      péntek
ITA      Venerd
MAC      petok
NLD      vrijdag
NOR      fredag
POL      pitek
PTG      Sexta-feira
/*      Server encoding of WCYRILLIC required for RUSDFWKX */
RUS      Пятница
SLO      petek
SVE      Fredag
```

xxxDFMNw.

Converts a numerical date into a form that contains the same calendar elements as MONNAME (that is, the name of the month), into a specific language/dialect.

xxx denotes the target language/dialect and is specified using one of:

- AFR for Afrikaans
- CAT for Catalan
- CRO for Croatian
- CSY for Czech
- DAN for Danish
- DES for Swiss_German
- DEU for German
- ENG for English
- ESP for Spanish
- FIN for Finnish
- FRA for French
- FRS for Swiss_French
- HUN for Hungarian
- ITA for Italian
- MAC for Macedonian
- NLD for Dutch
- NOR for Norwegian
- POL for Polish
- PTG for Portuguese
- RUS for Russian
- SLO for Slovenian

- SVE for Swedish

	min	max	default
Output widths for all languages/dialects except those listed below	1	32	8
Afrikaans	1	32	9
Danish	1	32	9
Swiss German	1	32	9
German	1	32	9
English	1	32	11
Spanish	1	32	10
Finnish	1	32	11
French	1	32	9
Swiss French	1	32	9
Hungarian	1	32	10
Italian	1	32	9
Macedonian	1	32	9
Dutch	1	32	9
Norwegian	1	32	9
Polish	1	32	12
Portuguese	1	32	9
Slovenian	1	32	9

The `EURDFMNw` format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, `DFLANG`.

Example

```
data _null_;  
d=0;  
put "AFR " d afrdfmn20.;  
put "CAT " d catdfmn20.;  
put "CRO " d crodfmn20.;  
put "CSY " d csydfmn20.;  
put "DAN " d dandfmn20.;  
put "DES " d desdfmn20.;  
put "DEU " d deudfmn20.;  
put "ENG " d engdfmn20.;  
put "ESP " d espdfmn20.;  
put "FIN " d findfmn20.;  
put "FRA " d fradfmn20.;  
put "FRS " d frsdfmn20.;  
put "HUN " d hundfmn20.;  
put "ITA " d itadfmn20.;  
put "MAC " d macdfmn20.;  
put "NLD " d nlddfmn20.;  
put "NOR " d nordfmn20.;  
put "POL " d poldfmn20.;  
put "PTG " d ptgdfmn20.;  
put "RUS " d rusdfmn20.;  
put "SLO " d slodfmn20.;  
put "SVE " d svedfmn20.;  
run;
```

Which produces the following output in the log:

```
AFR          Januarie  
CAT          Gener  
CRO          sijeanj  
CSY          leden  
DAN          januar  
DES          Januar  
DEU          Januar  
ENG          January  
ESP          enero  
FIN          tammikuuta  
FRA          janvier  
FRS          janvier  
HUN          január  
ITA          Gennaio  
MAC          januari  
NLD          januari  
NOR          januar  
POL          stycze  
PTG          janeiro  
/* Server encoding of WCYRILLIC required for RUSDFWKX */  
RUS          Янвaрь  
SLO          januar  
SVE          januari
```

xxxDFMYw.

Converts a numerical date into a form that contains the same calendar elements as MMMYY into a specific language/dialect.

The language/dialect code in the following table replaces xxx in the format name. For example, to convert to the default width Spanish, the format name is ESPDFDFMY5.

	language/dialect	min width	max width	default width
Afrikaans	AFR	5	7	5
Catalan	CAT	5	32	5
Croatian	CRO	5	32	5
Czech	CSY	10	32	10
Danish	DAN	5	7	5
Dutch	NLD	5	7	5
English	ENG	5	7	5
Finnish	FIN	8	8	8
French	FRA	5	7	5
Swiss French	FRS	5	7	5
German	DEU	5	7	5
Swiss German	DES	5	7	5
Hungarian	HUN	9	32	9
Italian	ITA	5	7	5
Macedonian	MAC	5	32	5
Norwegian	NOR	5	7	5
Polish	POL	5	32	5
Portuguese	PTG	5	7	5
Russian	RUS	5	32	5
Slovenian	SLO	5	32	5
Spanish	ESP	5	7	5
Swedish	SVE	5	7	5

The EURDFMYw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

Example

```
data _null_;  
d=0;  
put  "AFR " d afrdfmy.;  
put  "CAT " d catdfmy.;  
put  "CRO " d crodfmy.;  
put  "CSY " d csydfmy.;  
put  "DAN " d dandfmy.;  
put  "DES " d desdfmy.;  
put  "DEU " d deudfmy.;  
put  "ENG " d engdfmy.;  
put  "ESP " d espdfmy.;  
put  "FIN " d findfmy.;  
put  "FRA " d fradfmy.;  
put  "FRS " d frsdfmy.;  
put  "HUN " d hundfmy.;  
put  "ITA " d itadfmy.;  
put  "MAC " d macdfmy.;  
put  "NLD " d nlddfmy.;  
put  "NOR " d nordfmy.;  
put  "POL " d poldfmy.;  
put  "PTG " d ptgdfmy.;  
put  "RUS " d rusdfmy.;  
put  "SLO " d slodfmy.;  
put  "SVE " d svedfmy.;  
run;
```

Which produces the following output in the log:

```
AFR Jan60  
CAT Gen60  
CRO sij60  
CSY leden1960  
DAN jan60  
DES Jan60  
DEU Jan60  
ENG JAN60  
ESP ene60  
FIN tammi60  
FRA jan60  
FRS jan60  
HUN 1960.jan.  
ITA Gen60  
MAC jan60  
NLD jan60  
NOR jan60  
POL sty60  
PTG jan60  
/* Server encoding of WCYRILLIC required for RUSDFWKX */  
RUS ЯНБ1960  
SLO jan60  
SVE jan60
```

xxxDFWDXw.

Converts a numerical date into a form that contains the same calendar elements as WORDDATX, into a specific language/dialect.

The language/dialect code in the following table replaces xxx in the format name. For example, to convert to the default width Spanish, the format name is ESPDFWDX24.

	language/dialect	min width	max width	default width
Afrikaans	AFR	3	37	29
Catalan	CAT	3	40	16
Croatian	CRO	3	40	16
Czech	CSY	8	40	16
Danish	DAN	3	18	18
Dutch	NLD	2	38	28
English	ENG	3	32	28
Finnish	FIN	3	20	20
French	FRA	3	18	18
Swiss French	FRS	3	18	18
German	DEU	3	18	18
Swiss German	DES	3	18	18
Hungarian	HUN	6	40	18
Italian	ITA	3	17	17
Macedonian	MAC	3	40	17
Norwegian	NOR	3	17	17
Polish	POL	3	40	17
Portuguese	PTG	3	37	23
Russian	RUS	3	40	16
Slovenian	SLO	3	40	17
Spanish	ESP	3	24	24
Swedish	SVE	3	17	17

The EURDFWDXw. format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, DFLANG.

Example

```
data _null_;
d=0;
put "AFR " d afrdfwdx15.;
put "CAT " d catdfwdx15.;
```

```

put "CRO " d crodfwdx15.;
put "CSY " d csydfwdx25.;
put "DAN " d dandfwdx15.;
put "DES " d desdfwdx15.;
put "DEU " d deudfwdx15.;
put "ENG " d engdfwdx15.;
put "ESP " d espdfwdx15.;
put "FIN " d findfwdx15.;
put "FRA " d fradfwdx15.;
put "FRS " d frsdfwdx15.;
put "HUN " d hundfwdx15.;
put "ITA " d itadfwdx15.;
put "MAC " d macdfwdx15.;
put "NLD " d nlddfwdx15.;
put "NOR " d nordfwdx15.;
put "POL " d poldfwdx15.;
put "PTG " d ptgdfwdx25.;
put "RUS " d rusdfwdx15.;
put "SLO " d slodfwdx15.;
put "SVE " d svedfwdx15.;
run;

```

Which produces the following output in the log:

```

AFR      1 Jan 1960
CAT      1 Gen 1960
CRO      1. sij 1960
CSY      1 leden 1960
DAN      1. jan 1960
DES      1. Jan 1960
DEU      1. Jan 1960
ENG      1 Jan 1960
ESP      1 ene 1960
FIN      1. tammi 1960
FRA      1er jan 1960
FRS      1er jan 1960
HUN      1960.jan.1.
ITA      01 Gen 1960
MAC      1. jan 1960
NLD      1 jan 1960
NOR      1 jan 1960
POL      1 sty 1960
PTG      1 de janeiro de 1960
/* Server encoding of WCYRILLIC required for RUSDFWKX */
RUS      1 Янв 1960
SLO      1. jan 1960
SVE      1 jan 1960

```

xxxDFWKXw.

Converts a numerical date into a form that contains the same calendar elements as WEEKDATX, into a specific language/dialect.

The language/dialect code in the following table replaces xxx in the format name. For example, to convert to the default width Spanish, the format name is ESPDFWKX29.

	language/dialect	min	max	default
Variable width		3	40	29
Afrikaans	AFR	2	38	28
Catalan	CAT	2	40	27
Czech	CRO	2	40	25
Croatian	CSY	2	40	25
Danish	DAN	2	31	31
Dutch	NLD	7	40	16
English	ENG	3	37	37
Finnish	FIN	2	37	37
French	FRA	3	27	27
Swiss French	FRS	3	27	27
German	DEU	2	30	30
Swiss German	DES	2	30	30
Hungarian	HUN	3	40	28
Italian	ITA	3	28	28
Macedonian	MAC	3	40	29
Norwegian	NOR	3	26	26
Polish	POL	2	40	34
Portuguese	PTG	3	38	38
Russian	RUS	2	40	29
Slovenian	SLO	3	40	29
Spanish	ESP	1	35	35
Swedish	SVE	3	26	26

The `EURDFWKXw.` format does not indicate any specific language. Instead, it lets you set the language of the format by using a system option, `DFLANG`.

Example

```
data _null_;
d=0;
put "AFR " d afrdfwxx25.;
put "CAT " d catdfwxx25.;
put "CRO " d crodfwxx25.;
put "CSY " d csydfwxx25.;
put "DAN " d dandfwxx25.;
put "DES " d desdfwxx25.;
put "DEU " d deudfwxx25.;
put "ENG " d engdfwxx25.;
put "ESP " d espdfwxx25.;
put "FIN " d findfwxx25.;
put "FRA " d fradfwxx25.;
```

```

put "FRS " d frsdfwkw25.;
put "HUN " d hundfwkw25.;
put "ITA " d itadfwkw25.;
put "MAC " d macdfwkw25.;
put "NLD " d nlddfkwkw25.;
put "NOR " d nordfwkw25.;
put "POL " d poldfwkw25.;
put "PTG " d ptgdfwkw38.;
put "RUS " d rusdfwkw25.;
put "SLO " d slodfwkw25.;
put "SVE " d svedfwkw25.;
run;

```

Which outputs the following in the log:

```

AFR      Vrydag, 1 Jan 1960
CAT      Divendres, 1 Gen 1960
CRO      petak, 1. sij 1960
CSY      ptek, 1 leden 1960
DAN      fredag, den 1. jan 1960
DES      Freitag, 1. Jan 1960
DEU      Freitag, 1. Jan 1960
ENG      Friday, 1 Jan 1960
ESP      viernes, 1 ene 1960
FIN      Pe, 1. tammi 1960
FRA      Vendredi 1er jan 1960
FRS      Vendredi 1er jan 1960
HUN      1960.jan.1., péntek
ITA      Venerd, 01 Gen 1960
MAC      petok, 1. jan 1960
NLD      vrijdag, 1 jan 1960
NOR      fredag, 1 jan 1960
POL      pitek, 1 sty 1960
PTG      Sexta-feira, 1 de janeiro de 1960
/* Server encoding of WCYRILLIC required for RUSDFWKX */
RUS      Пятница, 1 Янв 1960
SLO      petek, 1. jan 1960
SVE      Fredag, 1 jan 1960

```

NLS-sensitive date formats

Formats that represent date-time data based on a specified locale value.

NLDATE_w.

Converts a date into a representation based upon the specified locale including the day number, the month and the year.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;
d=19995;
options locale=en_GB;
put d nldate.;
run;

data _null_;
d=19995;
options locale=fr_FR;
put d nldate.;
run;
```

Which produces the following output in the log:

```
29 September 2014
...
29 septembre 2014
```

NLDATEMDw.

Converts a date into a representation based upon the specified locale, writing it out as the day of the month followed by the name of the month.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;
d=19995;
options locale=en_GB;
put d nldatemd.;
run;

data _null_;
d=19995;
options locale=fr_FR;
put d nldatemd.;
run;
```

Which produces the following output in the log:

```
29 September
...
29 septembre
```

NLDATEMNw.

Converts a date into a representation based upon the specified locale, writing it out as the name of the month.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;  
d=19995;  
options locale=en_GB;  
put d nldatemn.;  
run;  
  
data _null_;  
d=19995;  
options locale=fr_FR;  
put d nldatemn.;  
run;
```

Which produces the following output in the log:

```
September  
...  
septembre
```

NLDATEWw.

Converts a date into a representation based upon the specified locale, writing the name of the day, the day of the month, the name of the month and the year.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;  
d=19995;  
options locale=en_GB;  
put d nldatew.;  
run;  
  
data _null_;  
d=19995;  
options locale=fr_FR;  
put d nldatew.;  
run;
```

Which produces the following output in the log:

```
Mon, 29 Sep 2014
...
lun. 29 sept. 2014
```

NLDATEWNw.

Converts a date into a representation based upon the specified locale, writing the name of the day.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;
d=19995;
options locale=en_GB;
put d nldatewn.;
run;

data _null_;
d=19995;
options locale=fr_FR;
put d nldatewn.;
run;
```

Which produces the following output in the log:

```
Monday
...
lundi
```

NLDATEYMw.

Converts a date into a representation based upon the specified locale, writing the name of the month and the year.

	min	max	default
Variable width	1	200	20

Example

```
data _null_;  
d=19995;  
options locale=en_GB;  
put d mandate.;  
run;  
  
data _null_;  
d=19995;  
options locale=fr_FR;  
put d mandate.;  
run;
```

Which produces the following output in the log:

```
September 2014  
...  
septembre 2014
```

NLDATEYRw.

Converts a date into a representation based upon the specified locale, just writing the year.

	min	max	default
Variable width	1	200	10

Example

```
data _null_;  
d=19995;  
options locale=en_GB;  
put d nldateyr.;  
run;  
  
data _null_;  
d=19995;  
options locale=fr_FR;  
put d nldateyr.;  
run;
```

Which produces the following output in the log:

```
2014  
...  
2014
```

NLDATEMw.

Converts a datetime into a representation based upon the specified locale, writing it as day number, month name, year and time.

	min	max	default
Variable width	1	50	30

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatm.;  
run;
```

```
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatm.;  
run;
```

Which produces the following output in the log:

```
29 September 2014 11:48:00  
...  
29 septembre 2014 11:48:00
```

NLDATEMAPw.

Converts a datetime into a representation based upon the specified locale, writing it as day number, month name, year, and time AM or PM.

	min	max	default
Variable width	1	50	30

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmap.;  
run;
```

```
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmap.;  
run;
```

Which produces the following output in the log:

```
29 September 2014 11:48:00 am
...
29 septembre 2014 11:48:00 AM
```

NLDATMDT w .

Converts a datetime into a representation based upon the specified locale, writing it as day number, month name and year.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;
d=1727610480;
options locale=en_GB;
put d nldatmdt.;
run;

data _null_;
d=1727610480;
options locale=fr_FR;
put d nldatmdt.;
run;
```

Which produces the following output in the log:

```
29 September 2014
...
29 septembre 2014
```

NLDATMMD w .

Converts a datetime into a representation based upon the specified locale, writing it as day number and the month name.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmmd.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmmd.;  
run;
```

Which produces the following output in the log:

```
29 September  
...  
29 septembre
```

NLDATMMN w .

Converts a datetime into a representation based upon the specified locale, writing the name of the month.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmmn.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmmn.;  
run;
```

Which produces the following output in the log:

```
September  
...  
septembre
```

NLDATMTMw.

Converts a datetime into a representation based upon the specified locale, writing the time of day.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;
d=1727610480;
options locale=en_GB;
put d nldatmtm.;
run;

data _null_;
d=1727610480;
options locale=fr_FR;
put d nldatmtm.;
run;
```

Which produces the following output in the log:

```
11:48:00
...
11:48:00
```

NLDATMTZw.

Converts a datetime into a representation based upon the specified locale, writing a time and a timezone.

	min	max	default
Variable width	1	50	30

Example

```
data _null_;
d=1727610480;
options locale=en_GB;
put d nldatmtz.;
run;

data _null_;
d=1727610480;
options locale=fr_FR;
put d nldatmtz.;
run;
```

Which produces the following output in the log:

```
11:48 BST
...
11:48 UTC+1
```

NLDATMW_w.

Converts a datetime into a representation based upon the specified locale, writing out a day, date number, year and time.

	min	max	default
Variable width	1	50	32

Example

```
data _null_;
d=1727610480;
options locale=en_GB;
put d nldatmw.;
run;

data _null_;
d=1727610480;
options locale=fr_FR;
put d nldatmw.;
run;
```

Which produces the following output in the log:

```
Mon, 29 Sep 2014 11:48
...
lundi 29 septembre 2014 11:48:00
```

NLDATMWZ_w.

Converts a datetime into a representation based upon the specified locale, writing out a day, date number, year, time and timezone.

	min	max	default
Variable width	1	50	36

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmwz.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmwz.;  
run;
```

Which produces the following output in the log:

```
Monday, 29 September 2014 11:48 BST  
...  
lundi 29 septembre 2014 11:48 UTC+1
```

NLDATMYMw.

Converts a datetime into a representation based upon the specified locale, writing the name of the month and a year.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmym.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmym.;  
run;
```

Which produces the following output in the log:

```
September 2014  
...  
septembre 2014
```

NLDATMYRw.

Converts a datetime into a representation based upon the specified locale, writing just the year.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmyr.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmyr.;  
run;
```

Which produces the following output in the log:

```
2014  
...  
2014
```

NLDATMZw.

Converts a datetime into a representation based upon the specified locale and including a time zone, writing it as shown below.

	min	max	default
Variable width	1	50	30

Example

```
data _null_;  
d=1727610480;  
options locale=en_GB;  
put d nldatmz.;  
run;  
  
data _null_;  
d=1727610480;  
options locale=fr_FR;  
put d nldatmz.;  
run;
```


Which produces the following output in the log:

```
29 September 2014 11:48 BST
...
29 septembre 2014 11:48 UTC+1
```

NLSTRMON w .

Converts a number representing the month of the year into a locale-specific string.

	min	max	default
Variable width	1	50	10

Example

```
data _null_;
d=1;
options locale=en_GB;
put d nlstrmon.;
run;

data _null_;
d=1;
options locale=fr_FR;
put d nlstrmon.;
run;
```

Which produces the following output in the log:

```
JAN
...
JANV.
```

NLSTRQTR w .

Converts a number representing an annual quarter into a locale-specific string.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1;  
options locale=en_GB;  
put d nlstrqtr.;  
run;  
  
data _null_;  
d=1;  
options locale=fr_FR;  
put d nlstrqtr.;  
run;
```

Which produces the following output in the log:

```
1st quarter  
...  
1er trimestre
```

NLSTRWKw.

Converts a number representing a day of the week into a locale-specific string.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=1;  
options locale=en_GB;  
put d nlstrwk.;  
run;  
  
data _null_;  
d=1;  
options locale=fr_FR;  
put d nlstrwk.;  
run;
```

Which produces the following output in the log:

```
Monday  
...  
lundi
```

NLTIMAP_w.

Converts a numeric time into a locale-specific string which includes a representation of AM or PM.

	min	max	default
Variable width	1	50	10

Example

```
data _null_;
d=3900;
options locale=en_GB;
put d nltimap20.;
run;

data _null_;
d=3900;
options locale=fr_FR;
put d nltimap20.;
run;
```

Which produces the following output in the log:

```
1:05:00 am
...
1:05:00 AM
```

NLTIME_w.

Converts a numeric time into a locale-relative string representation of the time.

	min	max	default
Variable width	1	50	10

Example

```
data _null_;
d=3900;
options locale=en_GB;
put d nltime.;
run;

data _null_;
d=3900;
options locale=fr_FR;
put d nltime.;
run;
```

Which produces the following output in the log:

```
01:05:00  
...  
01:05:00
```

NLS-sensitive money formats

Formats that represent numeric data as monetary values for a specified locale.

NLMNYw.d

Converts a numeric amount of money into a money-format representation in the locale currency.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=10.51;  
options locale=en_US;  
put d nlmny9.2;  
run;  
  
data _null_;  
d=10.51;  
options locale=fr_FR;  
put d nlmny9.2;  
run;
```

Which produces the following output in the log:

```
$10.51  
...  
10,51 €
```

NLMNYlw.d

Converts a numeric amount of money into a money-format representation containing a string locale currency code.

	min	max	default
Variable width	1	50	20

Example

```
data _null_;  
d=10.51;  
options locale=en_US;  
put d nlmnyi9.2;  
run;  
  
data _null_;  
d=10.51;  
options locale=fr_FR;  
put d nlmnyi9.2;  
run;
```

Which produces the following output in the log:

```
USD10.51  
...  
10,51 EUR
```

NLMNIxxxw.d

Formats a numeric input as a currency; the final appearance of the value to display is informed by the session locale.

xxx in the format denotes the currency, and is specified using one of:

- AED UAE Dirham
- AUD Australia Dollar
- BGN Bulgaria Lev
- BRL Brazil Real
- CAD Canada Dollar
- CHF Switzerland Franc
- CNY China Yuan Renminbi
- CZK Czech Republic Koruna
- DKK Denmark Krone
- EGP Egypt Pound
- EUR Euro
- GBP Great Britain Pound
- HKD Hong Kong Dollar
- HRK Croatia Kuna
- HUF Hungary Forint
- IDR Indonesia Rupiah
- ILS Israel Shekel

- INR India Rupee
- JPY Japan Yen
- KRW South Korea Won
- LTL Lithuania Litas
- LVL Latvia Lat
- MOP Macau Pataca
- MXN Mexico Peso
- MYR Malaysia Ringgit
- NOK Norway Krone
- NZD New Zealand Dollar
- PLN Poland Zloty
- RUB Russia, Ruble
- SEK Sweden Krona
- SGD Singapore Dollar
- THB Thailand Baht
- TRY Turkey Lira
- TWD Taiwan New Dollar
- USD United States Dollar
- ZAR South Africa Rand

	min	max	default
Variable width	1	50	20

Example

```
data _null_;
options locale=en_GB;
d = 10.51;
put d nlmniaed10.2;
put d nlmniaud10.2;
put d nlmnibgn10.2;
put d nlmnibr110.2;
put d nlmnicad10.2;
put d nlmnichf10.2;
put d nlmnicny10.2;
put d nlmniczk10.2;
put d nlmnidkk10.2;
put d nlmniegp10.2;
put d nlmnieur10.2;
put d nlmnigbp10.2;
put d nlmnihkd10.2;
put d nlmnihrk10.2;
put d nlmnihuf10.2;
put d nlmniidr10.2;
put d nlmniils10.2;
put d nlmniinr10.2;
```

```
put d nlmnijpy10.2;  
put d nlmnikrw10.2;  
put d nlmniltl10.2;  
put d nlmnilvl10.2;  
put d nlmnimop10.2;  
put d nlmnimxn10.2;  
put d nlmnimyr10.2;  
put d nlmninok10.2;  
put d nlmminzd10.2;  
put d nlmnipln10.2;  
put d nlmnirub10.2;  
put d nlmnisek10.2;  
put d nlmnisgd10.2;  
put d nlmnithb10.2;  
put d nlmnitry10.2;  
put d nlmnitwd10.2;  
put d nlmniusd10.2;  
put d nlmnizar10.2;  
run;
```

Which produces the following output in the log:

```
AED10.51  
AUD10.51  
BGN10.51  
BRL10.51  
CAD10.51  
CHF10.51  
CNY10.51  
CZK10.51  
DKK10.51  
EGP10.51  
EUR10.51  
GBP10.51  
HKD10.51  
HRK10.51  
HUF10.51  
IDR10.51  
ILS10.51  
INR10.51  
JPY10.51  
KRW10.51  
LTL10.51  
LVL10.51  
MOP10.51  
MXN10.51  
MYR10.51  
NOK10.51  
NZD10.51  
PLN10.51  
RUB10.51  
SEK10.51  
SGD10.51  
THB10.51  
TRY10.51  
TWD10.51  
USD10.51  
ZAR10.51
```

NLMNLxxxw.d

Converts the numeric input into a local form for a currency; the final appearance of the value to display is informed by the session locale.

xxx in the format denotes the currency, and is specified using one of:

- AED UAE Dirham
- AUD Australia Dollar
- BGN Bulgaria Lev
- BRL Brazil Real
- CAD Canada Dollar
- CHF Switzerland Franc
- CNY China Yuan Renminbi
- CZK Czech Republic Koruna
- DKK Denmark Krone
- EGP Egypt Pound
- EUR Euro
- GBP Great Britain Pound
- HKD Hong Kong Dollar
- HRK Croatia Kuna
- HUF Hungary Forint
- IDR Indonesia Rupiah
- ILS Israel Shekel
- INR India Rupee
- JPY Japan Yen
- KRW South Korea Won
- LTL Lithuania Litas
- LVL Latvia Lat
- MOP Macau Pataca
- MXN Mexico Peso
- MYR Malaysia Ringgit
- NOK Norway Krone
- NZD New Zealand Dollar
- PLN Poland Zloty
- RUB Russia, Ruble
- SEK Sweden Krona
- SGD Singapore Dollar

- THB Thailand Baht
- TRY Turkey Lira
- TWD Taiwan New Dollar
- USD United States Dollar
- ZAR South Africa Rand

	min	max	default
Variable width	1	50	20

Example

```
data _null_;
options locale=en_GB;
d = 10.51;
put d nlmnlaed10.2;
put d nlmnlaud10.2;
put d nlmnlbgn10.2;
put d nlmnlbrl10.2;
put d nlmnlcad10.2;
put d nlmnlchf10.2;
put d nlmnlcny10.2;
put d nlmnlczk10.2;
put d nlmnldkk10.2;
put d nlmnlegp10.2;
put d nlmnleur10.2;
put d nlmnlgbp10.2;
put d nlmnlhkd10.2;
put d nlmnlhrk10.2;
put d nlmnlhuf10.2;
put d nlmnlidr10.2;
put d nlmnlils10.2;
put d nlmnlinr10.2;
put d nlmnljpy10.2;
put d nlmnlkrw10.2;
put d nlmnlltl10.2;
put d nlmnllvl10.2;
put d nlmnlmop10.2;
put d nlmnlmxn10.2;
put d nlmnlmyr10.2;
put d nlmnlnok10.2;
put d nlmnlnoz10.2;
put d nlmnlpln10.2;
put d nlmnlrub10.2;
put d nlmnlsek10.2;
put d nlmnlsgd10.2;
put d nlmnlthb10.2;
put d nlmnltry10.2;
put d nlmnltwd10.2;
put d nlmnlusd10.2;
put d nlmnlzar10.2;
run;
```

Which produces the following output in the log:

```
AED10.51
AU$10.51
BGN10.51
R$10.51
CA$10.51
CHF10.51
CN¥10.51
CZK10.51
DKK10.51
EGP10.51
€10.51
£10.51
HK$10.51
HRK10.51
HUF10.51
IDR10.51
₪10.51
#10.51
¥10.51
₩10.51
LTL10.51
LVL10.51
MOP10.51
MX$10.51
MYR10.51
NOK10.51
NZ$10.51
PLN10.51
RUB10.51
SEK10.51
SGD10.51
#10.51
TRY10.51
NT$10.51
$10.51
ZAR10.51
```

YENw.d

Converts a numeric amount into a corresponding representation in the Yen currency.

	min	max	default
Variable width	2	32	6

NLS-sensitive numeric formats

Formats that represent numeric data according to a specified locale.

NLNUMw.d

Converts its numeric input into a number formatted using the session locale's number format, in which the decimal place and the 'thousands' separators are determined by the locale itself.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;
options locale=en_GB;
d = 1025.989;
put d nlnum8.3;
run;
```

```
data _null_;
options locale=fr_FR;
d = 1025.989;
put d nlnum8.3;
run;
```

```
data _null_;
options locale=de_DE;
d = 1025.989;
put d nlnum8.3;
run;
```

Which produces the following output in the log:

```
1,025.99
...
1 026,0
...
1.025,99
```

NLNUMlw.d

Converts its numeric input into an international number format, using commas and periods as separators.

This format differs from the `NLNUM` format, in which the separators are locale-determined. It also differs from the `COMMA` format, which rounds more aggressively (losing precision) when attempting to accommodate smaller output widths.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;
options locale=en_GB;
d = 1025.989;
put d nlnumi9.3;
run;

data _null_;
options locale=fr_FR;
d = 1025.989;
put d nlnumi9.3;
run;

data _null_;
options locale=de_DE;
d = 1025.989;
put d nlnumi9.3;
run;
```

Which produces the following output in the log:

```
1,025.989
...
1,025.989
...
1,025.989
```

NLPCTw.d

Converts its numeric input into a percentage using the locale's number format.

	min	max	default
Variable width	1	32	6

Example

The second example below demonstrates how the number 1.7767 is expressed as a percentage using the French locale – by using a comma as the decimal point and placing a space before the percentage symbol.

```
data _null_;
options locale=en_GB;
d = 1.7767;
put d nlpct10.2;
run;

data _null_;
options locale=fr_FR;
d = 1.7767;
put d nlpct10.2;
run;
```

Which produces the following output in the log:

```
177.67%  
...  
177,67 %
```

NLPCTIw.d

Converts its numeric input into a percentage form using an international number format, in which the decimal point is represented by a period, but some locale-specific conventions are preserved (for example, the insertion of a space before the percentage symbol in certain locales).

	min	max	default
Variable width	1	32	6

Example

```
data _null_;  
options locale=en_GB;  
d = 1.7767;  
put d nlpcti10.2;  
run;  
  
data _null_;  
options locale=fr_FR;  
d = 1.7767;  
put d nlpcti15.2;  
run;  
  
data _null_;  
options locale=de_DE;  
d =1.7767;  
put d nlpcti10.2;  
run;
```

Which produces the following output in the log:

```
177.67%  
...  
177.67 %  
...  
177.67 %
```

NLPCTN $w.d$

Converts its numeric input into a percentage, prepending a minus sign to negative values.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;  
d = 1.7767;  
e = -1.7767;  
put d nlpctn10.2;  
put e nlpctn10.2;  
run;
```

Which produces the following output in the log:

```
177.67%  
-177.67%
```

NLPCTP $w.d$

Converts its numeric input into a percentage form using locale specific separators for thousands and the decimal point.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;  
options locale=en_GB;  
d = 1.7767;  
e = -1.7767;  
put d nlpctp10.2;  
put e nlpctp10.2;  
run;  
  
data _null_;  
options locale=fr_FR;  
d = 1.7767;  
e = -1.7767;  
put d nlpctp10.2;  
put e nlpctp10.2;  
run;
```

Which produces the following output in the log:

```
177.67%  
-177.67%  
...  
177,67%  
-177,67%
```

NLBEST $w.d$

Converts its numeric input into the best locale-specific representation.

	min	max	default
Variable width	1	32	12

Example

```
data _null_;  
options locale=en_GB;  
d = 1111.7767;  
e = -1111.7767;  
put d nlbest10.2;  
put e nlbest10.2;  
run;  
  
data _null_;  
options locale=fr_FR;  
d = 1111.7767;  
e = -1111.7767;  
put d nlbest10.2;  
put e nlbest10.2;  
run;
```

Which produces the following output in the log:

```
1111.7767  
-1111.7767  
...  
1111,7767  
-1111,7767
```

NLPVALUE $w.d$

Converts its numeric input into a p-value represented according to the session locale.

	min	max	default
Variable width	3	32	6

Example

```
data _null_;
options locale=en_GB;
d = 1.11;
e = 0.11;
put d nlpvalue10.2;
put e nlpvalue10.2;
run;

data _null_;
options locale=fr_FR;
d = 1.11;
e = 0.11;
put d nlpvalue10.2;
put e nlpvalue10.2;
run;
```

Which produces the following output in the log:

```
1.11
0.11
...
1,11
0,11
```

Informats

Informats define display representation of the input dataset.

Informats are used when you need to acquire formatted, non-standard data and bring it into a SAS language program. Their function is suggested in the name - *in*format – they facilitate formatted input, essentially informing WPS how to read variables. The SAS language only supports two types of data – standard character and standard numeric – but it is possible that you may wish to process external data that isn't presented as either standard character or standard numeric. This is what informats are for, and they are often employed as part of `INPUT` statements. Informats are usually classified into three broad categories:

Character informats

Character informats transform external character data into a standard character form. Character informats have the pattern: `$<informat-name>w.` - `w` is the total width of the input field and the terminating period is mandatory. The `$w.` informat, which omits an informat name, is used for reading standard character data values.

Numeric informats

These transform external numeric data into a standard numeric form. Numeric informats have the pattern: `<informat-name>w.d` - `w` is the total width of the input field and `d` is an optional power of 10 by which the result is divided (but only if the input does not contain a decimal point, in which case it is ignored). The period following the width is, again, mandatory. The `w.d` informat, which omits a name, is used for reading standard numeric data.

Date informats

These transform character representations of dates into a standard form - a number representing the number of days since the SAS language epoch date - 1st Jan 1960. Closely-related, datetime formats perform similar services, returning the number of elapsed seconds since 00:00:00 on the epoch date. Date and datetime informats have the pattern: \$<informat-name>w. - w is the total width of the input field and the terminating period is mandatory.

Example informat use cases

The following example uses the \$w. informat to read character data - by default, it trims leading blanks and left-aligns the input:

```
data _null_;
input s $20.;
put s;
cards;
    World Programming
;
run;
```

The output:

```
World Programming
```

Next, the w.d informat is used to read standard numeric data - note how the second item has been divided by 10,000:

```
data _null_;
input s 9.4;
put s;
cards;
123456.78
12345678
;
run;
```

Which produces the output:

```
123456.78
1234.5678
```

Finally, an example of reading non-standard character data - a representation of a date. The ANYDTDTE. informat converts a variety of date-like and time-like values into a numeric date value

```
data _null_;
input d anydtdte11.;
put d;
cards;
29 SEP 2014
;
run;
```

This informat has transformed a non-standard representation of a date into a standard, numeric, epoch-based value:

```
19995
```

Core informats

Some of the most widely used informats, fundamental to the SAS language.

BESTw.d

This informat reads numeric data. Trailing blanks on input data are ignored and its default length is 12. It is identical to Dw.d. and Ew.d.

Example

```
data _null_;  
input s best7.2;  
put s;  
cards;  
123456  
1234.56  
;  
run;
```

Which produces the following output in the log:

```
1234.56  
1234.56
```

Dw.d

This informat reads numeric data. Trailing blanks on input data are ignored and its default length is 12. It is identical to BESTw.d. and Ew.d.

Example

```
data _null_;  
input s d7.2;  
put s;  
cards;  
123456  
1234.56  
;  
run;
```

Which produces the following output in the log:

```
1234.56  
1234.56
```

Ew.d

This informat reads numeric data. Trailing blanks on input data are ignored and its default length is 12. It is identical to BESTw.d. and Dw.d.

Example

```
data _null_;  
input s e7.2;  
put s;  
cards;  
123456  
1234.56  
;  
run;
```

Which produces the following output in the log:

```
1234.56  
1234.56
```

Fw.d

This format reads external numeric data – trailing blanks on input data are ignored.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
input s f7.2;  
put s;  
cards;  
123456  
1234.56  
;  
run;
```

Which produces the following output in the log:

```
1234.56  
1234.56
```

Basic character informats

Fundamental character informats for different platforms.

\$ASCIIw.

This informat reads ASCII data into a native-format string. On an ASCII platform, this has no effect.

	min	max	default
Variable width	1	32767	1

\$BASE64Xw.

This informat reads base64-encoded ASCII text into character data.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;
s="World Programming";
t=put(s, $base64x100.); /* Convert to base64 */
put t;
r=input(t, $base64x100.); /* Read in from base64 */
put r;
run;
```

Which produces the following output in the log:

```
V29ybGQgUHJvZ3JhbW1pbmc=
World Programming
```

\$BINARYw.

This informat reads a string containing a sequence of binary characters into a character representation.

	min	max	default
Variable width	1	32767	8,*8

Example

```
data _null_;
s="010101110101000001010011";
r=input(s, $binary24.);
put r;
run;
```

Which produces the following output in the log:

```
WPS
```

\$CHARw.

This informat reads character data into a string. If enough space is available, leading and trailing blanks are retained.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s="   World Programming   ";  
r=input(s, $char30.);  
put r $30. "*";  
run;
```

Which produces the following output in the log:

```
World Programming   *
```

\$CHARZBw.

This informat reads hex-encoded character data into a string. Binary zeroes are converted to blanks.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;  
s="0000575053"x;  
r=input(s, $charzb20.);  
put r $20.;  
run;
```

Which produces the following output in the log:

```
WPS
```

\$CSTRw.

This informat reads a null-terminated string into a blank-padded string.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;
s="57505300"x;
r=input(s,$cstr8.);
put r "*";
put r $hex. "*";
run;
```

Which produces the following output in the log:

```
WPS *
57505320202020*
```

\$EBCDICw.

On an EBCDIC platform, this informat has no effect. On an ASCII platform, it transforms EBCDIC input data into an ASCII representation.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;
s="WPS";
r=put(s,$ebcdic20.);
t=input(r, $ebcdic20.);
put t;
run;
```

Which produces the following output in the log:

```
WPS
```

\$Fw.

This informat reads character data into a string, trimming leading and trailing blanks.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;  
s="          World Programming          ";  
t=input(s, $f100.);  
put t "*";  
run;
```

Which produces the following output in the log:

```
World Programming *
```

\$HEXw.

This informat reads hex-encoded character data into a session-encoded string.

	min	max	default
Variable width	1	32767	2,*2

Example

```
data _null_;  
s="575053";  
r=input(s, $hex20.);  
put r;  
run;
```

Which produces the following output in the log:

```
WPS
```

\$PHEXw.

This informat reads packed hexadecimal character data into a session-encoded string.

	min	max	default
Variable width	1	32767	2,/2

\$QUOTEw.

This informat reads character data, removing pairs of quotation marks.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s=' "WPS" ' ;  
r=input(s,$quote.);  
put r;  
put s;  
run;
```

Which produces the following output in the log:

```
WPS  
"WPS"
```

\$REVERJw.

This informat reads character data into a visually reversed form, retaining leading and trailing blanks.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;  
s="      gnimmargorP dlroW      ";  
r=input(s,$reverj30.);  
put r $30. "*";  
run;
```

Which produces the following output in the log:

```
World Programming      *
```


\$REVERS w .

This informat reads character data into a visually reversed form, stripping trailing blanks first, so that the output is left-aligned.

	min	max	default
Variable width	1	32767	1

Example

```
data _null_;  
s="      gnimmargorP dlroW      ";  
r=input(s,$revers30.);  
put r $30. "*";  
run;
```

Which produces the following output in the log:

```
World Programming      *
```

\$UPCASE w .

This informat reads character data into an upper case form.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s="world programming";  
r=input(s,$upcase30.);  
put r;  
run;
```

Which produces the following output in the log:

```
WORLD PROGRAMMING
```

\$VARYINGw.

This informat reads character strings of varying length. Its second parameter sets the maximum length of the variable being processed.

Example

```
data _null_ ;
  length c $ 15 ;
  input len 2. c $varying15. len date date9. ;
  put c date;
cards ;
  7777777777nov2014
  1211111111111111oct2014
;
```

Which produces the following output in the log:

```
7777777 20034
11111111111 20007
```

Bidirectional informats

Character informats that deal with data written from left to right and from right to left.

\$LOGVS w .

This informat reads a string in left-to-right logical order into a string in visual order.

	min	max	default
Variable width	1	32767	200

Example

```
data _null_;
s='#####';
r=input(s, $logvs20.);
put r;
run;
```

Which produces the following output in the log:

```
/* System encoding needs to be set to ARABIC for correct behaviour */
#####
```

\$LOGVSRw.

This informat reads a string in right-to-left logical order into a string in visual order.

	min	max	default
Variable width	1	32767	200

Example

```
data _null_;  
s="#####";  
r=input(s, $logvsr20.);  
put r;  
run;
```

Which produces the following output in the log:

```
/* System encoding needs to be set to ARABIC for correct behaviour */  
#####
```

\$VSLOGw.

This informat reads a string in visual order into a string in left-to-right logical order.

	min	max	default
Variable width	1	32767	200

Example

```
data _null_;  
s="#####";  
r=input(s, $vslog20.);  
put r;  
run;
```

Which produces the following output in the log:

```
/* System encoding needs to be set to ARABIC for correct behaviour */  
#####
```

\$VSLOGRw.

This informat reads a string in visual order into a string in right-to-left logical order.

	min	max	default
Variable width	1	32767	200

Example

```
data _null_;  
s='#####';  
r=input(s, $vslogr20.);  
put r;  
run;
```

Which produces the following output in the log:

```
/* System encoding needs to be set to ARABIC for correct behaviour */  
#####
```

Unicode informats

Character informats for different variants of the Unicode encoding.

\$UCS2Bw.

This informat reads data in big-endian, 16-bit UCS2 Unicode encoding into a session-encoded character data form.

	min	max	default
Variable width	2	32000	8

Example

```
data _null_;  
s="0057006F0072006C0064002000500072006F006700720061006D006D0069006E0067"x;  
r=input(s, $ucs2b60.);  
put r;  
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UCS2BEw.

This informat reads session-encoded character strings into a big-endian, 16-bit UCS2 Unicode encoding.

	min	max	default
Variable width	2	32000	8

Example

```
data _null_;  
s="World Programming";  
r=input(s, $ucs2be60.);  
put r;  
run;
```

Which produces the following output in the log:

```
W o r l d   P r o g r a m m i n g
```

\$UCS2Lw.

This informat reads little-endian, 16-bit UCS2 Unicode encoded data into session-encoded character form.

	min	max	default
Variable width	2	32000	8

Example

```
data _null_;  
s="57006F0072006C0064002000500072006F006700720061006D006D0069006E006700"x;  
r=input(s, $ucs2l60.);  
put r;  
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UCS2LEw.

This informat reads a session-encoded character string into a little-endian 16-bit UCS2 form.

	min	max	default
Variable width	2	32000	8

Example

```
data _null_;  
s="World Programming";  
r=input(s, $ucs2le100.);  
put r;  
run;
```

Which produces the following output in the log:

```
W o r l d   P r o g r a m m i n g
```

\$UCS2Xw.

This informat reads 16-bit UCS2 machine-endian unicode data into a character string.

	min	max	default
Variable width	2	32000	8

Example

```
/* If executed on an x86 machine */  
data _null_;  
s="57006F0072006C0064002000500072006F006700720061006D006D0069006E006700"x;  
r=input(s, $ucs2x100.);  
put r;  
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UCS2XEW.

This informat reads session-encoded character data into a 16-bit UCS2 Unicode encoding in the endianness of the executing machine.

	min	max	default
Variable width	2	32000	8

Example

```
data _null_;
s="World Programming";
r=input(s, $ucs2xe100.);
put r;
run;
```

Which produces the following output in the log:

```
/* If executed on an x86 machine */
W o r l d   P r o g r a m m i n g
World Programming
```

\$UCS4BEW.

This informat reads big-endian, 32-bit UCS4 Unicode encoded data into a session-encoded character form.

	min	max	default
Variable width	4	32000	8

Example

```
data _null_;
s="0000005700000006F000000720000006C00000064000000200000005000000072
0000006F00000006700000072000000610000006D0000006D000000690000006E00000067"x;
t=input(s, $ucs4b100.);
put t;
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UCS4Lw.

This informat reads little-endian, 32-bit UCS4 Unicode-encoded data into a session-encoded character form.

	min	max	default
Variable width	4	32000	8

Example

```
data _null_;
s="5700000006F000000720000006C00000064000000200000005000000072000000
6F0000006700000072000000610000006D0000006D000000690000006E00000067000000"x;
t=input(s, $ucs4l100.);
put t;
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UCS4Xw.

This informat converts 32-bit UCS4 Unicode machine-endian encoded data into a session-encoded character form.

	min	max	default
Variable width	4	32000	8

Example

```
/* If executed on an x86 machine */
data _null_;
s="5700000006F000000720000006C000000640000002000000050000000720000006F000000
6700000072000000610000006D0000006D000000690000006E00000067000000"x;
t=input(s, $ucs4x100.);
put t;
run;
```

Which produces the following output in the log:

```
World Programming
```


\$UCS4XEW.

This informat reads a session-encoded character string into machine-endian 32-bit UCS4 Unicode encoded data.

	min	max	default
Variable width	4	32000	8

Example

```
data _null_;  
s="World Programming";  
t=input(s, $ucs4xe100.);  
put t;  
run;
```

Which produces the following output in the log:

```
/* If executed on an x86 machine */  
W   o   r   l   d       P   r   o   g   r   a   m   m   i   n   g
```

\$UESCW.

This informat reads a UESC character string into a session-encoded character string.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s="\u0023World Programming\u0024";  
t=input(s, $uesc50.);  
put t;  
run;
```

Which produces the following output in the log:

```
#World Programming$
```

\$UESCEw.

This informat reads character data converting all but the 0–9, A–Z, a–z and space characters to Unicode universal character names in the \uXXXX notation.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s="#World Programming$";  
t=input(s, $uesce50.);  
put t;  
run;
```

Which produces the following output in the log:

```
\u0023World Programming\u0024
```

\$UNCRw.

This informat reads character data, converting characters in the Unicode numeric character reference format (&#dddd; notation) into a session-encoded character form. This notation is described in <http://www.w3.org/TR/html4/charset.html#h-5.3.1>

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;  
s="#&#00035;World Programming&#00038;";  
t=input(s, $uncr100.);  
put t;  
run;
```

Which produces the following output in the log:

```
#World Programming&
```

\$UNCREw.

This informat reads character data, converting all but 0–9, A–Z, a–z and space to the Unicode numeric character reference format (&#dddd notation). This notation is described in <http://www.w3.org/TR/html4/charset.html#h-5.3.1> [↗](#).

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;
s="&World Programming#";
t=input(s, $uncre50.);
put t;
run;
```

Which produces the following output in the log:

```
&#00038;World Programming&#00035;
```

\$UPARENw.

This informat reads a sequence of characters encoded individually as <uxxxx> (where xxxx is the Unicode code point for the character in hexadecimal) into a session-encoded string.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;
s="<u0057><u006F><u0072><u0063><u0064><u0020><u0050><u0072><u006F><u0067><u0072><u0061><u0064><u0064><u0069><u0065><u0067>";
t=input(s, $uparen200.);
put t;
run;
```

Which produces the following output in the log:

```
World Programming
```

\$UPARENEw.

This informat reads a session-encoded string into a string representation in which each character is encoded as <uxxxx> where xxxx is the Unicode code point for the character concerned.

	min	max	default
Variable width	1	32767	8

Example

```
data _null_;
s="World Programming";
t=input(s, $uparene200.);
put t;
run;
```

Which produces the following output in the log:

```
<u0057><u006F><u0072><u006C><u0064><u0020><u0050><u0072><u006F>
<u0067><u0072><u0061><u006D><u006D><u0069><u006E> 1t;u0067>
```

\$UTF8Xw.

This informat transforms UTF-8 data into session-encoded data.

	min	max	default
Variable width	1	32000	8

\$UTF8XEW.

This informat transforms session-encoded data into UTF-8.

	min	max	default
Variable width	1	32000	8

Simple numeric informats

Fundamental informats for numeric data.

BINARYw.d

This informat reads a binary value into a numeric value.

	min	max	default
Variable width	1	64	8

Example

```
data _null_;  
d=3;  
b=put(d, binary10.);  
put b;  
r=input(b, binary10.);  
put r;  
run;
```

Which produces the following output in the log:

```
0000000011  
3
```

BITSw.d

This informat reads a specified number of bits from its argument, converting them into a numeric variable. The number of bits is determined by the width and the process starts at an offset determined by the specified number of decimal digits.

	min	max	default
Variable width	1	64	1

Example

```
data _null_;  
d="Z"; /* ASCII 90 = BINARY 01011010 */  
r=input(d, bits4.1);  
put r;  
run;
```

Which produces the following output in the log:

```
11
```

BZw.d

This informat reads numeric values converting spaces to zeroes. The decimal part, if present, denotes a power of 10 to divide the value provided.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
d="1000";  
r=input(d, bz6.2);  
put r;  
run;
```

Which produces the following output in the log:

```
10
```

COMMAw.d

This informat reads numeric data, removing embedded commas and currency signs. The decimal part, if present, denotes a power of 10 to divide the value provided.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
d="123,456,789";  
r=input(d, comma10.3);  
put r;  
run;
```

Which produces the following output in the log:

```
12345.678
```

COMMAXw.d

This informat reads numeric data, removing embedded periods and currency signs. The decimal part, if present, denotes a power of 10 to divide the value provided.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
d="123.456.789";  
r=input(d, commax10.3);  
put r;  
run;
```

Which produces the following output in the log:

```
12345.678
```

DOLLARw.d

This informat reads numeric data, removing embedded commas and currency signs. The decimal part, if present, denotes a power of 10 to divide the value provided.

This informat is an alias of COMMAw.d

	min	max	default
Variable width	1	32	1

DOLLARXw.d

This informat reads numeric data, removing embedded commas and currency signs. The decimal part, if present, denotes a power of 10 to divide the value provided.

This informat is an alias of COMMAw.d

	min	max	default
Variable width	1	32	1

EUROW.d

This informat reads numeric data removing embedded Euro currency symbols, commas and other characters.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
s=1096543.123;  
t= put(s, euro20.3);  
r= input(t, euro20.3);  
put t;  
put r;  
run;
```

Which produces the following output in the log:

```
E1,096,543.123  
1096543.123
```

EUROX.d

This informat reads numeric data removing embedded Euro currency characters.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
s="E1096543.123";  
r= input(s, eurox20.3);  
put r;  
run;
```

Which produces the following output in the log:

```
1096543.123
```


FLOAT*w.d*

This informat reads a floating point value. If present, the decimal part represents a power of 10 by which the result is divided.

	min	max	default
Variable width	4	4	4

Example

```
data _null_;  
s=123.456;  
t=put(s,float4.);  
r=input(t, float4.2);  
put r;  
run;
```

Which produces the following output in the log:

```
1.2345600128
```

HEX*w.*

This informat reads hexadecimal character data into a numeric form.

	min	max	default
Variable width	1	16	8

Example

```
data _null_;  
s="FF";  
r=input(s, hex.);  
put r;  
run;
```

Which produces the following output in the log:

```
255
```

IBw.d

This informat reads integer binary data, converting to session form.

	min	max	default
Variable width	1	8	4

IBRw.d

This informat reads integer binary data in a machine-specific form and converts it to a session form.

	min	max	default
Variable width	1	8	4

NUMXw.d

This informat reads numeric values converting commas to periods.

	min	max	default
Variable width	1	32	12

Example

```
data _null_;
s=123456.789;
t=put(s, numx12.4);
put t;
r=input(t, numx12.);
put r;
run;
```

Which produces the following output in the log:

```
123456,7890
123456.789
```

PDw.d

This informat reads packed decimal data into a numeric value.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;  
s=123;  
t=put(s, pd3.);  
put t $hex.;  
r=input(t, pd3.);  
put r;  
run;
```

Which produces the following output in the log:

```
000123  
123
```

PERCENTw.d

This informat reads percentages into a numeric value.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;  
s=0.28;  
t=put(s, percent.);  
put t;  
r=input(t, percent.);  
put r;  
run;
```

Which produces the following output in the log:

```
28%  
0.28
```

PIBw.d

This informat reads a positive integer binary representation into a numeric value.

	min	max	default
Variable width	1	8	1

Example

```
data _null_;  
s="88D612"x;  
r=input(s,pib8.0);  
put r;  
run;
```

Which produces the following output in the log:

```
1234568
```

PIBRw.d

This informat reads a little-endian positive integer representation into a numeric value.

	min	max	default
Variable width	1	8	1

Example

```
data _null_;  
s="88D612"x;  
r=input(s,pibr8.0);  
put r;  
run;
```

Which produces the following output in the log:

```
1234568
```

PKw.d

This informat reads a packed decimal representation into a numeric form.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;  
s=1234567.95;  
r=put(s, pk7.4);  
put r $hex.;  
t=input(r, pk7.4);  
put t;  
run;
```

Which produces the following output in the log:

```
00012345679500  
1234567.95
```

RBw.d

This informat reads a real binary representation into a numeric value.

	min	max	default
Variable width	2	8	4

Example

```
data _null_;  
s=1234567.95;  
r=put(s, rb4.0);  
put r $hex.;  
t=input(r, rb5.0);  
put t;  
run;
```

Which produces the following output in the log:

```
87D63241  
1234567
```

S370FFw.d

This informat reads open edition 1047 EBCDIC (IBM mainframe) format on non-z/OS machines into a numeric value. It has no effect on z/OS machines.

	min	max	default
Variable width	1	32	12

Example

```
data _null_;  
s=10;  
r=put(s, s370ff.);  
put r $hex.;  
t=input(r, s370ff.);  
put t;  
run;
```

Which produces the following output in the log:

```
4040404040404040404040F1F0  
10
```

S370FIBw.d

This informat reads big-endian integer-binary IBM mainframe format data into a numeric form.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;  
s=10;  
r=put(s, s370fib8.);  
put r $hex.;  
t=input(r, s370fib8.);  
put t;  
run;
```

Which produces the following output in the log:

```
0000000000000000A  
10
```

S370FIBUw.d

This informat reads unsigned big-endian positive integer binary IBM mainframe format into a numeric value.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;
s=10;
r=put(s, s370fibu8.);
put r $hex.;
t=input(r, s370fibu8.);
put t;
run;
```

Which produces the following output in the log:

```
0000000000000000A
10
```

S370FPDw.d

This informat reads z/OS packed decimal format data into a numeric value.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;
s=10;
r=put(s, s370fpd8.);
put r $hex.;
t=input(r,s370fpd8.);
put t;
run;
```

Which produces the following output in the log:

```
0000000000000010C
10
```

S370FPDUw.d

This informat reads z/OS unsigned packed decimal format into a numeric value.

	min	max	default
Variable width	1	16	1

Example

```
data _null_;
s=10;
r=put(s, s370fpdu8.);
put r $hex.;
t=input(r,s370fpdu8.);
put t;
run;
```

Which produces the following output in the log:

```
0000000000000010F
10
```

S370FPIB*w.d*

This informat reads z/OS big-endian positive integer binary format into a numeric form.

	min	max	default
Variable width	1	8	4

Example

```
data _null_;
s=10;
r=put(s, s370fpib8.);
put r $hex.;
t=input(r,s370fpib8.);
put t;
run;
```

Which produces the following output in the log:

```
0000000000000000A
10
```

S370FRB*w.d*

This informat reads z/OS real binary format into a numeric value.

	min	max	default
Variable width	2	8	6

Example

```
data _null_;
s=10;
r=put(s, s370frb8.);
put r $hex.;
t=input(r,s370frb8.);
put t;
run;
```

Which produces the following output in the log:

```
41A0000000000000
10
```

S370FZD*w.d*

This informat reads z/OS zoned decimal format into a numeric value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
s=10;
r=put(s, s370fzd8.);
put r $hex.;
t=input(r,s370fzd8.);
put t;
run;
```

Which produces the following output in the log:

```
F0F0F0F0F0F0F1C0
10
```

S370FZDB*w.d*

This informat reads z/OS zoned decimal format into a numeric value.

	min	max	default
Variable width	1	32	8

S370FZDLw.d

This informat reads z/OS zoned decimal format with a sign nibble at the beginning into a numeric form.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;  
s=10;  
r=put(s, s370fzdl8.);  
put r $hex.;  
t=input(r,s370fzdl8.);  
put t;  
run;
```

Which produces the following output in the log:

```
C0F0F0F0F0F0F1F0  
10
```

S370FZDSw.d

This informat reads z/OS zoned decimal format with a sign byte at the beginning into a numeric value.

	min	max	default
Variable width	2	32	8

Example

```
data _null_;  
s=10;  
r=put(s, s370fzds8.);  
put r $hex.;  
t=input(r,s370fzds8.);  
put t;  
run;
```

Which produces the following output in the log:

```
4EF0F0F0F0F0F1F0  
10
```

S370FZDTw.d

This informat reads z/OS zoned decimal format with a sign byte at the end into a numeric form.

	min	max	default
Variable width	2	32	8

Example

```
data _null_;
s=10;
r=put(s, s370fzdt8.);
put r $hex.;
t=input(r,s370fzdt8.);
put t;
run;
```

Which produces the following output in the log:

```
F0F0F0F0F0F0F1F04E
10
```

S370FZDUw.d

This informat reads z/OS zoned decimal format with no sign byte into a numeric value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
s=10;
r=put(s, s370fzdu8.);
put r $hex.;
t=input(r,s370fzdu8.);
put t;
run;
```

Which produces the following output in the log:

```
F0F0F0F0F0F0F1F0
10
```

TRAILSGN*w.d*

This informat reads a character representation of a number with a trailing sign into a numeric value. If present, the decimal part represents a power of 10 by which the result is divided.

	min	max	default
Variable width	1	32	6

Example

```
data _null_;  
s="123+";  
t="456-";  
r1=input(s, trailsgn10.0);  
r2=input(t, trailsgn10.4);  
put r1;  
put r2;  
run;
```

Which produces the following output in the log:

```
123  
-0.0456
```

YEN*w.d*

This informat reads numeric data, removing embedded commas and yen signs. The decimal part, if present, denotes a power of 10 to divide the value provided.

	min	max	default
Variable width	1	32	1

ZD*w.d*

This informat reads a platform-dependent zoned decimal format into a numeric value.

	min	max	default
Variable width	1	32	1

Example

```
data _null_;  
s=123456;  
r=put(s,zd10.2);  
put r $hex.;  
t=input(r,zd10.2);  
put t;  
run;
```

Which produces the following output in the log:

```
3030313233343536307B  
123456
```

ZDBw.d

This informat converts platform-dependent zoned decimal data in which zeros are blank into a numeric value.

	min	max	default
Variable width	1	32	1

Numeric date informats

Informats that convert a numeric date into a formatted character representation.

ANYDTDTEw.

This informat reads a variety of date-like and time-like values into a numeric date value.

	min	max	default
Variable width	5	60	9

Example

```
data _null_;  
input checkdate : anydtdte.;  
put checkdate;  
put checkdate=: date9.;  
cards;  
18SEP2014  
18SEP2014 12:24:32.8  
SEP2014  
;  
run;
```

Which produces the following output in the log:

```
19984  
checkdate=18SEP2014  
19984  
checkdate=18SEP2014  
19967  
checkdate=01SEP2014
```

ANYDTDTMw.

This informat reads a range of date-like and time-like values into a numeric datetime value.

	min	max	default
Variable width	1	60	19

Example

```
data _null_;  
input checkdatetime : anydtdtm20.;  
put checkdatetime;  
put checkdatetime=: datetime25.;  
cards;  
18SEP2014  
18SEP2014 12:24:32.8  
SEP2014  
;  
run;
```

Which produces the following output in the log:

```
1726617600  
checkdatetime=18SEP2014:00:00:00  
1726617600  
checkdatetime=18SEP2014:00:00:00  
1725148800  
checkdatetime=01SEP2014:00:00:00
```

ANYDTTMEw.

This informat reads a variety of time-like values into a numeric time value.

	min	max	default
Variable width	1	60	8

Example

```
data _null_;  
input checktime : anydttme20.;  
put checktime;  
put checktime=: time12.;  
cards;  
12:24:32.8  
;  
run;
```

Which produces the following output in the log:

```
44672.8  
checktime=12:24:33
```

DATEw.

This informat reads date values in the form DDMMYY or DDMMYYYY or DD-MMM-YYYY into a numeric date.

	min	max	default
Variable width	7	32	7

Example

```
data _null_;  
e="01-JAN-1960";  
d="18-SEP-2014";  
r1=input(e,date20.);  
r2=input(d,date20.);  
put r1;  
put r2;  
run;
```

Which produces the following output in the log:

```
0  
19984
```

DATETIMEw.

This informat reads datetime values in the form DDMMYY:HH:MM:SS or DDMMYYYY:HH:MM:SS or DD-MMM-YYYY:HH:MM:SS into a numeric datetime.

	min	max	default
Variable width	13	40	18

Example

```
data _null_;  
e="01JAN60:00:00:00";  
d="18-SEP-14:15:27:20";  
r1=input(e,datetime20.);  
r2=input(d,datetime20.);  
put r1;  
put r1 datetime20.;  
put r2;  
put r2 datetime20.;  
run;
```

Which produces the following output in the log:

```
0  
    01JAN1960:00:00:00  
1726673240  
    18SEP2014:15:27:20
```

DDMMYYw.

This informat reads a date in a DD/MM/YY or DD/MM/YYYY representation into a numeric date.

	min	max	default
Variable width	6	32	6

Example

```
data _null_;  
d="19/09/2014";  
r=input(d, ddmmyy20.);  
put r;  
run;
```

Which produces the following output in the log:

```
19985
```


HHMMSS_w.

This informat reads a time expressed as HH:MM:SS into a numeric time format.

	min	max	default
Variable width	1	20	8

Example

```
data _null_;  
d="09:43:00";  
r=input(d, hhmmss20.);  
put r;  
run;
```

Which produces the following output in the log:

```
34980
```

JULIAN_w.

This informat reads a YYDDD or YYYYDDD Julian date into a numeric format.

	min	max	default
Variable width	5	32	5

Example

```
data _null_;  
d="2014034";  
r=input(d, julian20.);  
put r;  
run;
```

Which produces the following output in the log:

```
19757
```

MDYAMP_w.

This informat reads datetimes expressed as MM-DD-YY HH.MM AM|PM or MM-DD-YYYY HH.MM AM|PM into a numeric datetime form.

	min	max	default
Variable width	8	40	16

Example

```
data _null_;  
d="09-15-14 4.45 pm";  
r=input(d,mdyampm20.);  
put r;  
run;
```

Which produces the following output in the log:

```
1726418700
```

MINGUOW.

This informat reads a Taiwanese date form into a numeric date.

	min	max	default
Variable width	6	10	6

Example

```
data _null_;  
d="49/01/01";  
e="0490101";  
r=input(d,minguo10.);  
s=input(e,minguo10.);  
put r;  
put s;  
run;
```

Which produces the following output in the log:

```
0  
0
```

MMDDYYw.

This informat reads a date in a MM/DD/YY or MM/DD/YYYY representation into a numeric date.

	min	max	default
Variable width	6	32	6

Example

```
data _null_;
d="09/19/2014";
r=input(d,mmddy10.);
put r;
run;
```

Which produces the following output in the log:

```
19985
```

MONYYw.

This informat converts a date expressed as MMMYY or MMMYYYY into a numeric date format.

	min	max	default
Variable width	5	32	5

Example

```
data _null_;
d="SEP2014";
r=input(d,monyy20.);
put r;
run;
```

Which produces the following output in the log:

```
19967
```

MSECw.

MSEC reads the output of the z/OS TIME macro, or equivalently the STCK instruction.

The STCK instruction produces a 64 bit big-endian value where bit position 51 (where the first bit is position 0) is incremented every microsecond. It returns the value as a time of day.

	min	max	default
Variable width	1	8	8

NENGOW.

This informat reads a Japanese date format (including an initial era signifier) into a numeric date.

	min	max	default
Variable width	10	16	10

Example

```
data _null_;  
d="H.26/08/15";  
r=input(d,nengo10.);  
put r;  
run;
```

Which produces the following output in the log:

```
19950
```

PDJULGW.

This informat reads a packed decimal representation of a Julian date into a numeric date.

	min	max	default
Variable width	3	4	4

Example

```
data _null_;  
d=0;  
r=put(d,pdjulg4.); /* Convert date to packed decimal form */  
put r $hex.; /* Examine it */  
s=input(r,pdjulg4.); /* Read and format it as (original) date */  
put s;  
run;
```

Which produces the following output in the log:

```
1960001F  
0
```

PDJULIw.

This informat reads a packed decimal representation (CCYYDDDF) of a Julian date into a numeric date.

	min	max	default
Variable width	4	4	4

Example

```
data _null_;  
d=0;  
r=put(d,pdjuli4.);  
put r $hex.;  
s=input(r,pdjuli4.);  
put s;  
run;
```

Which produces the following output in the log:

```
0060001F  
0
```

PDTIMEw.

This informat reads a packed decimal representation of a time into a numeric time.

	min	max	default
Variable width	4	4	4

Example

```
data _null_;  
d="0112825F"x;  
r=input(d,ptime4.);  
put r;  
put r $time.;  
run;
```

Which produces the following output in the log:

```
41305  
11:28:25
```

RMFDUR_w.

Reads durations stored in z/OS RMF (Resource Management Facility) records.

	min	max	default
Variable width	4	4	4

RMFSTAMP_w.

Reads time values stored in z/OS RMF (Resource Management Facility) records.

	min	max	default
Variable width	8	8	8

SHRSTAMP_w.

Reads timestamps from z/OS SHR records.

	min	max	default
Variable width	8	8	8

SMFSTAMP_w.

Reads timestamps from z/OS SMF records.

	min	max	default
Variable width	8	8	8

TIME_w.

Reads a representation of a time in HH:MM:SS format into a numeric time.

	min	max	default
Variable width	5	32	8

Example

```
data _null_;
t="01:01:01";
r=input(t,time9.);
put r;
run;
```

Which produces the following output in the log:

```
3661
```

TODSTAMP w .

Reads the output of the z/OS STCK instruction and produces a datetime value, rather than a time of day value.

	min	max	default
Variable width	1	8	8

TU w .

Reads the output of the z/OS TIME macro with the TU option.

This informat produces a time of day as an unsigned 32 bit number. The low order bit is equal to one timer unit. There are exactly 38,400 timer units per second, so a timer unit is approximately 26.041667 microseconds.

	min	max	default
Variable width	4	4	4

YMDDTTM $w.d$

Reads datetimes in the form YY-MM-DD HH:SS or YYYY-MM-DD HH:SS into numeric datetime objects.

	min	max	default
Variable width	13	40	18

Example

```
data _null_;  
d="2014-09-19 14:33";  
r=input(d, ymddttm20.);  
put r;  
run;
```

Which produces the following output in the log:

```
1726756380
```

YYMMDDw.

Reads dates in the form YY-MM-DD or YYYY-MM-DD into numeric date objects.

	min	max	default
Variable width	6	32	6

Example

```
data _null_;  
d="2014-09-19";  
r=input(d, yymmdd20.);  
put r;  
run;
```

Which produces the following output in the log:

```
19985
```

YYMMNw.

Reads dates in the form YYYYMM or YYMM into numeric date objects.

	min	max	default
Variable width	4	6	4

Example

```
data _null_;  
d="201409";  
r=input(d, yymmn6.);  
put r;  
run;
```


Which produces the following output in the log:

```
19967
```

YYQw.

Reads yearly quarters in the form YYQn or YYYYQn into a numeric date, where n is 1,2,3 or 4.

	min	max	default
Variable width	4	32	6

Example

```
data _null_;  
d="2014Q1";  
r=input(d, yyq6.);  
put r;  
run;
```

Which produces the following output in the log:

```
19724
```

ISO8601 date informats

Informats that represent date-time data according to the ISO 8601 standard.

\$N8601Bw.d

Reads ISO 8601 datetimes, durations and intervals expressed in basic or extended form.

	min	max	default
Variable width	1	32767	50

Example

```
data _null_;  
input d $n8601b.;  
put d;  
cards;  
P20140915T134200  
;  
run;
```

Which produces the following output in the log:

```
2014915134200FFD
```

\$N8601Ew.d

Reads ISO 8601 datetimes, durations and intervals expressed in extended form only.

	min	max	default
Variable width	1	32767	50

Example

```
data _null_;  
d="P2014-09-15T13:42:00"; /* Extended notation */  
conv = input(d, $n8601e.);  
put conv;  
run;
```

Which produces the following output in the log:

```
2014915134200FFD
```

B8601DAw.

Converts a basic ISO 8601 YYYYMMDD form into a numeric date.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d="19600101";  
conv=input(d,b8601da.);  
put conv;  
run;
```

Which produces the following output in the log:

```
0
```

B8601DNw.

Reads a basic ISO 8601 YYYYMMDD form into a numeric datetime in which the time component is set to zero.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d="19600101";  
conv=input(d,b8601dn.);  
put conv;  
run;
```

Which produces the following output in the log:

```
0
```

B8601DTw.

Converts a basic ISO 8601 YYYYMMDDTHHMMSS form into a numeric datetime.

	min	max	default
Variable width	19	26	19

Example

```
data _null_;  
dt="19600101T000001";  
conv=input(dt,b8601dt.);  
put conv;  
run;
```

Which produces the following output in the log:

```
1
```

B8601DZw.

Reads a UTC ISO 8601 YYYYMMDDTHHMMSS+|=HHMM datetime and timezone form into a numeric datetime.

	min	max	default
Variable width	20	35	26

Example

```
data _null_;  
dt="19600101T000000+0000";  
conv=input(dt,b8601dz.);  
put conv;  
run;
```

Which produces the following output in the log:

```
0
```

B8601LZw.

Converts an input in ISO 8601 basic UTC time notation: HHMMSS+|-HHMM to a numeric time.

This informat is an alias of B8601TZ.

	min	max	default
Variable width	9	20	14

B8601TMw.

Reads a basic ISO 8601 time form: HHMMSS into a numeric time.

	min	max	default
Variable width	6	15	8

Example

```
data _null_;  
t="010000";  
conv=input(t,b8601tm.);  
put conv;  
run;
```

Which produces the following output in the log:

```
3600
```

B8601TZw.

Converts an input in ISO 8601 basic UTC time notation: HHMMSS+|-HHMM to a numeric time.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
t="152001-0500";  
conv=input(t,b8601tz20.20);  
put conv;  
run;
```

Which produces the following output in the log:

```
73201
```

E8601DAw.

Reads an extended ISO 8601 YYYY-MM-DD form into a numeric date.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d="2014-09-18";  
e="1960-01-01";  
r=input(d, E8601DA.);  
s=input(e, E8601DA.);  
put r;  
put s;
```

Which produces the following output in the log:

```
19984  
0
```

E8601DNw.

Reads an extended ISO 8601 YYYY-MM-DD form into a numeric datetime.

	min	max	default
Variable width	10	10	10

Example

```
data _null_;  
d="2014-09-18";  
e="1960-01-01";  
r=input(d, E8601DN.);  
s=input(e, E8601DN.);  
put r;  
put s;
```

Which produces the following output in the log:

```
1726617600  
0
```

E8601DTw.

Reads an extended ISO 8601 YYYY-MM-DDTHH:MM:SS datetime form into a numeric datetime.

	min	max	default
Variable width	19	26	19

Example

```
data _null_;  
d="2014-09-17T09:25:00";  
e="1960-01-01T00:00:00";  
r=input(d, E8601DT.);  
s=input(e, E8601DT.);  
put r;  
put s;  
run;
```

Which produces the following output in the log:

```
1726565100  
0
```

E8601DZw.

Reads an extended UTC ISO 8601 YYYY-MM-DDTHH:MM:SS+|=HH:MM datetime and timezone form into a numeric datetime.

	min	max	default
Variable width	20	35	26

Example

```
data _null_;  
d="2014-09-17T09:25:00+09:00";  
e="1960-01-01T00:00:00+00:00";  
r=input(d, E8601DZ.);  
s=input(e, E8601DZ.);  
put r;  
put s;  
run;
```

Which produces the following output in the log:

```
1726532700  
0
```

E8601TMw.

Reads an extended ISO 8601 form: HH:MM:SS into a numeric time.

	min	max	default
Variable width	8	15	8

Example

```
data _null_;  
d="06:45:00";  
r=input(d, E8601TM.);  
put r;  
run;
```

Which produces the following output in the log:

```
24300
```

E8601TZw.

Reads an ISO 8601 extended time notation HH:MM:SS+|-HH:MM form into a numeric time.

	min	max	default
Variable width	9	20	14

Example

```
data _null_;  
d="06:45:00+01:15";  
r=input(d, E8601TZ.);  
put r;  
run;
```

Which produces the following output in the log:

```
19800
```

E8601LZw.

Reads an ISO 8601 extended time notation HH:MM:SS+|-HH:MM form into a numeric time.

This informat is an alias of E8601TZ.

	min	max	default
Variable width	9	20	14

IS8601DAw.

Reads an extended ISO 8601 YYYY-MM-DD form into a numeric date.

This informat is an alias of E8601DA.

	min	max	default
Variable width	10	10	10

IS8601DNw.

Reads an extended ISO 8601 YYYY-MM-DD form into a numeric datetime.

This informat is an alias of E8601DN.

	min	max	default
Variable width	10	10	10

IS8601DTw.

Reads an extended ISO 8601 YYYY-MM-DDTHH:MM:SS datetime form into a numeric datetime.

This informat is an alias of `E8601DT`.

	min	max	default
Variable width	19	26	19

IS8601DZw.

Reads an extended UTC ISO 8601 YYYY-MM-DDTHH:MM:SS+|=HH:MM datetime and timezone form into a numeric datetime.

This informat is an alias of `E8601DZ`.

	min	max	default
Variable width	20	35	26

IS8601LZw.

Reads an ISO 8601 extended time notation HH:MM:SS+|-HH:MM form into a numeric time.

This informat is an alias of `E8601LZ`.

	min	max	default
Variable width	9	20	14

IS8601TMw.

Reads an extended ISO 8601 form: HH:MM:SS into a numeric time.

This informat is an alias of `E8601TM`.

	min	max	default
Variable width	8	15	8

IS8601TZ_w.

Reads an ISO 8601 extended time notation HH:MM:SS+|-HH:MM form into a numeric time.

This informat is an alias of `E8601TZ`.

	min	max	default
Variable width	9	20	14

ND8601DA_w.

Reads an extended ISO 8601 YYYY-MM-DD form into a numeric date.

This informat is an alias of `B8601DA`.

	min	max	default
Variable width	10	10	10

ND8601DN_w.

Reads a basic ISO 8601 YYYYMMDD form into a numeric datetime in which the time component is set to zero.

This informat is an alias of `B8601DN`.

	min	max	default
Variable width	10	10	10

ND8601DT_w.

Converts a basic ISO 8601 YYYYMMDDTHHMMSS form into a numeric datetime.

This informat is an alias of `B8601DT`.

	min	max	default
Variable width	19	26	19

ND8601DZw.

Reads an extended UTC ISO 8601 YYYY-MM-DDTHH:MM:SS+|=HH:MM datetime and timezone form into a numeric datetime.

This informat is an alias of B8601DZ.

	min	max	default
Variable width	20	35	26

ND8601LZw.

Converts an input in ISO 8601 basic UTC time notation: HHMMSS+|-HHMM to a numeric time.

This informat is an alias of B8601LZ.

	min	max	default
Variable width	9	20	14

ND8601TMw.

Reads a basic ISO 8601 time form: HHMMSS into a numeric time.

This informat is an alias of B8601TM.

	min	max	default
Variable width	6	15	8

ND8601TZw.

Converts an input in ISO 8601 basic UTC time notation: HHMMSS+|-HHMM to a numeric time.

This is an alias of B8601TZ.

	min	max	default
Variable width	9	20	14

NLS-sensitive date informats

Informats that represent date-time data based on a specified locale value.

NLDATEw.

Reads a locale-specific date representation into a numeric date value.

	min	max	default
Variable width	1	32	12

Example

```
options locale=en_GB;
data _null_;
d=input("23 September 2014", nldate19.);
put d=;
run;

options locale=fr_FR;
data _null_;
e=input("23 septembre 2014", nldate19.);
put e=;
run;
```

Which produces the following output in the log:

```
d=19989
...
e=19989
```

NLDATMw.

Reads a locale-specific datetime representation into a numeric datetime.

	min	max	default
Variable width	1	50	20

Example

```
options locale=en_GB;
data _null_;
d=input("23 September 2014 11:06:00", nldatm26.);
put d=;
run;

options locale=fr_FR;
data _null_;
e=input("23 septembre 2014 11:06:00", nldatm26.);
put e=;
run;
```

Which produces the following output in the log:

```
d=1727089560
...
e=1727089560
```

NLTIMEw.

Reads a locale-specific time representation into a time value.

	min	max	default
Variable width	1	32	10

Example

```
options locale=en_GB;
data _null_;
d=input("11:06:00", nltime26.);
put d=;
run;

options locale=fr_FR;
data _null_;
e=input("11:06:00", nltime26.);
put e=;
run;
```

Which produces the following output in the log:

```
d=39960
...
e=39960
```

NLS-sensitive money informats

Informats that represent numeric data as monetary values for a specified locale.

NLMNY*w.d*

Reads a locale-specific money-format representation into a numeric amount of money.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
options locale=en_US;
am="$10.51";
r=input(am, nlmny.);
put r;
run;
```

Which produces the following output in the log:

```
10.51
```

NLMNYI*w.d*

Reads a money-format representation containing a string locale currency code into a numeric money value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
options locale=en_US;
am="USD10.51";
r=input(am, nlmnyi.);
put r;
run;
```

Which produces the following output in the log:

```
10.51
```

NLMNIXXXw.d

Reads a country-specific currency form into a numeric amount.

xxx in the format denotes the currency, and is specified using one of:

- AED UAE Dirham
- AUD Australia Dollar
- BGN Bulgaria Lev
- BRL Brazil Real
- CAD Canada Dollar
- CHF Switzerland Franc
- CNY China Yuan Renminbi
- CZK Czech Republic Koruna
- DKK Denmark Krone
- EGP Egypt Pound
- EUR Euro
- GBP Great Britain Pound
- HKD Hong Kong Dollar
- HRK Croatia Kuna
- HUF Hungary Forint
- IDR Indonesia Rupiah
- ILS Israel Shekel
- INR India Rupee
- JPY Japan Yen
- KRW South Korea Won
- LTL Lithuania Litas
- LVL Latvia Lat
- MOP Macau Pataca
- MXN Mexico Peso
- MYR Malaysia Ringgit
- NOK Norway Krone
- NZD New Zealand Dollar
- PLN Poland Zloty
- RUB Russia, Ruble
- SEK Sweden Krona
- SGD Singapore Dollar

- THB Thailand Baht
- TRY Turkey Lira
- TWD Taiwan New Dollar
- USD United States Dollar
- ZAR South Africa Rand

	min	max	default
Variable width	1	32	8

Example

```
data _null_;  
am="AED10.51";  
r=input(am, nlmniaed.);  
put r;  
run;
```

Which produces the following output in the log:

```
10.51
```

NLMNLXXXw.d

Reads a local form of a country-specific currency into a numeric amount.

xxx in the format denotes the currency, and is specified using one of:

- AED UAE Dirham
- AUD Australia Dollar
- BGN Bulgaria Lev
- BRL Brazil Real
- CAD Canada Dollar
- CHF Switzerland Franc
- CNY China Yuan Renminbi
- CZK Czech Republic Koruna
- DKK Denmark Krone
- EGP Egypt Pound
- EUR Euro
- GBP Great Britain Pound
- HKD Hong Kong Dollar
- HRK Croatia Kuna
- HUF Hungary Forint

- IDR Indonesia Rupiah
- ILS Israel Shekel
- INR India Rupee
- JPY Japan Yen
- KRW South Korea Won
- LTL Lithuania Litas
- LVL Latvia Lat
- MOP Macau Pataca
- MXN Mexico Peso
- MYR Malaysia Ringgit
- NOK Norway Krone
- NZD New Zealand Dollar
- PLN Poland Zloty
- RUB Russia, Ruble
- SEK Sweden Krona
- SGD Singapore Dollar
- THB Thailand Baht
- TRY Turkey Lira
- TWD Taiwan New Dollar
- USD United States Dollar
- ZAR South Africa Rand

	min	max	default
Variable width	1	32	8

Example

```
data _null_;  
am="€10.51";  
r=input(am, nlmnleur.);  
put r;  
run;
```

Which produces the following output in the log:

```
10.51
```

NLS-sensitive numeric informats

Informats that represent numeric data according to a specified locale.

NLNUM*w.d*

Reads locale-specific numeric data into a numeric form.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;  
options locale="en_GB";  
am="1,254,234";  
r=input(am, nlnum32.);  
put r;  
run;
```

Which produces the following output in the log:

```
1254234
```

NLNUMI*w.d*

Reads locale-specific international format numbers into a numeric value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;  
options locale="en_GB";  
am="1,254,234";  
r=input(am, nlnumi32.);  
put r;  
run;
```

Which produces the following output in the log:

```
1254234
```

NLPCTw.d

Reads a locale-specific percentage expression into a numeric value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
options locale="en_GB";
am="10 %";
r=input(am, nlpct3.2);
put r;
run;
```

Which produces the following output in the log:

```
0.1
```

NLPCTI

Reads an international, session-locale percentage into a numeric value.

	min	max	default
Variable width	1	32	8

Example

```
data _null_;
options locale="en_GB";
am="10 %";
r=input(am, nlpcti3.2);
put r;
run;
```

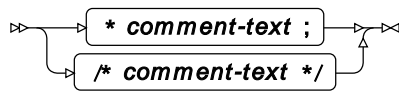
Which produces the following output in the log:

```
0.1
```

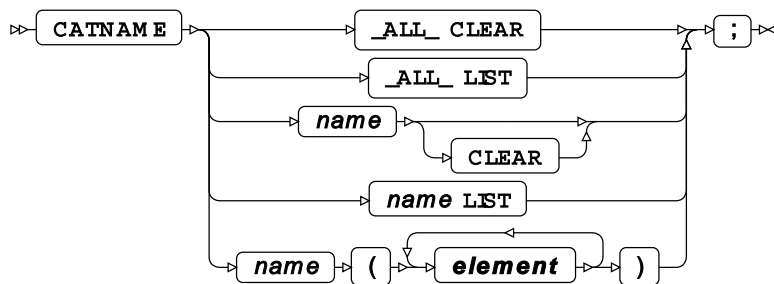
Global statements

Statements that can be used anywhere in a SAS language program

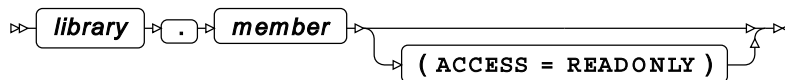
Comment



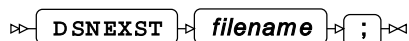
CATNAME



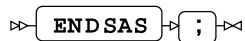
element



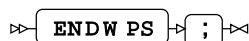
DSNEXST



ENDSAS

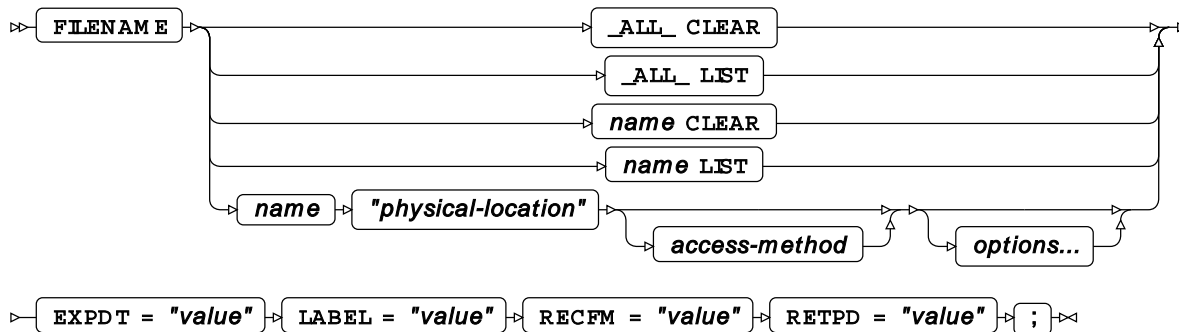


ENDWPS



FILENAME statements

FILENAME

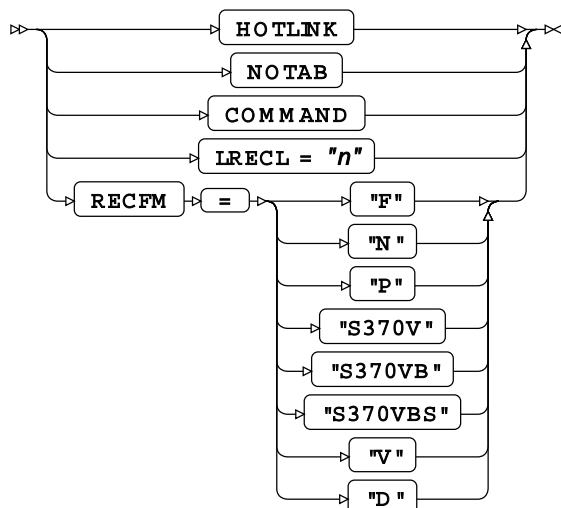


On z/OS only, a further `ENGINE=` option is supported to specify the `INFILE/FILE` user exit that will be used when accessing the file reference through the `INFILE/FILE` statements in a DATA step.

FILENAME, DDE Access Method



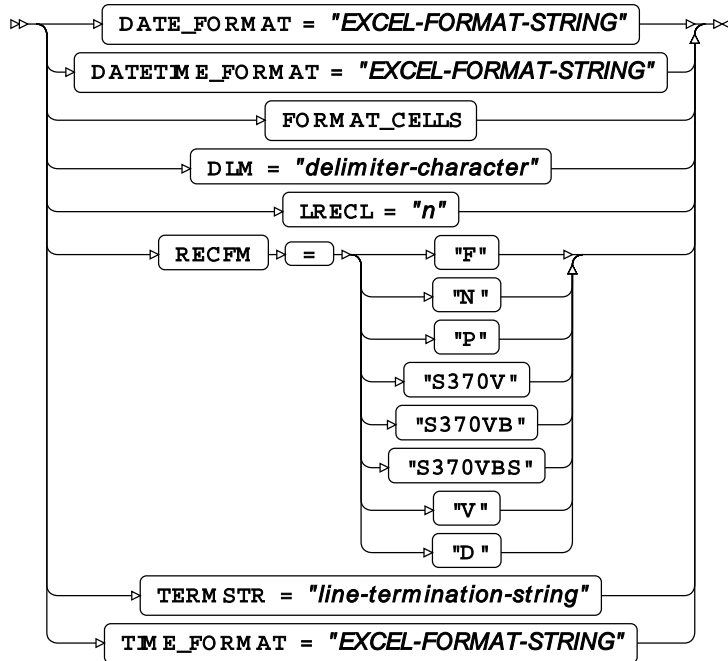
dde-option



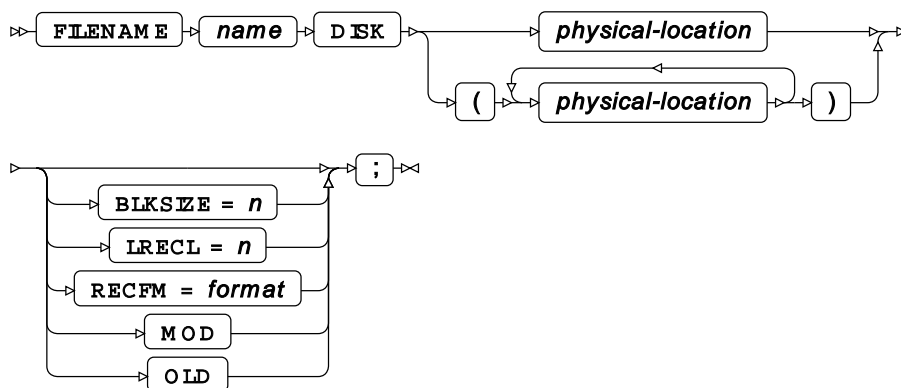
FILENAME, DDEX Access Method



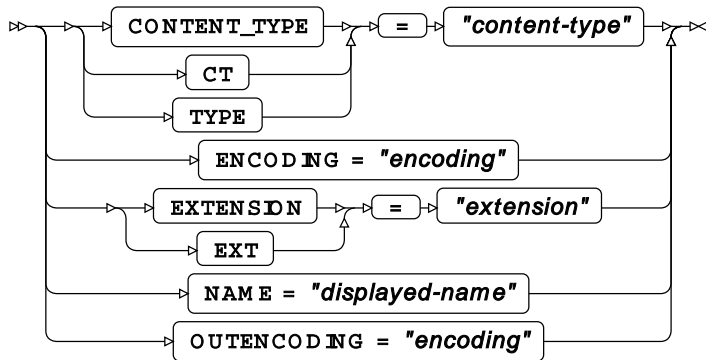
ddex-option



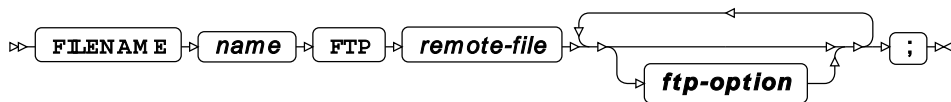
FILENAME, DISK Access Method



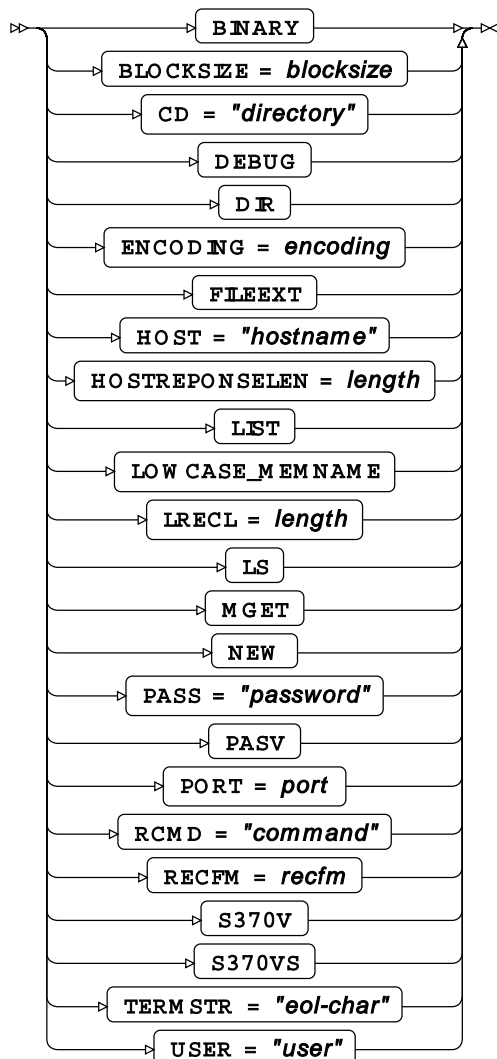
attachment-option



FILENAME, FTP Access Method



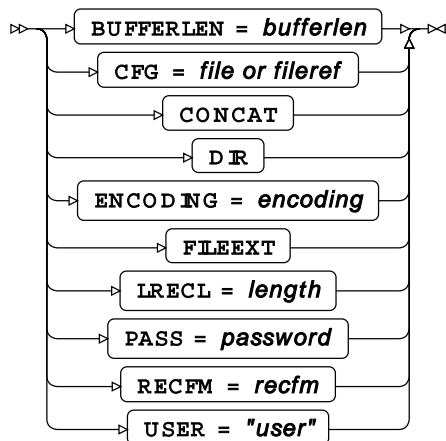
ftp-option



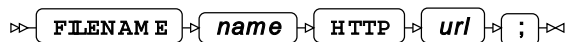
FILENAME, HADOOP Access Method



hadoop-option



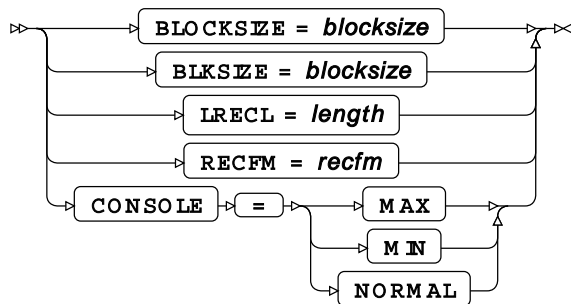
FILENAME, HTTP Access Method



FILENAME, PIPE Access Method



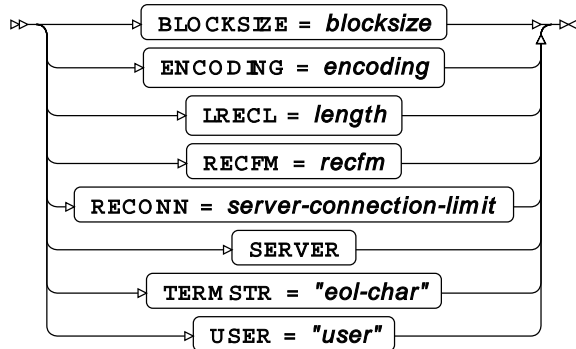
pipe-option



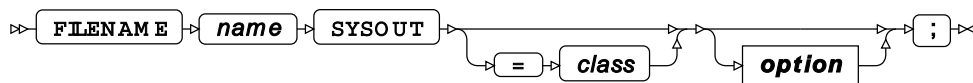
FILENAME, SOCKET Access Method



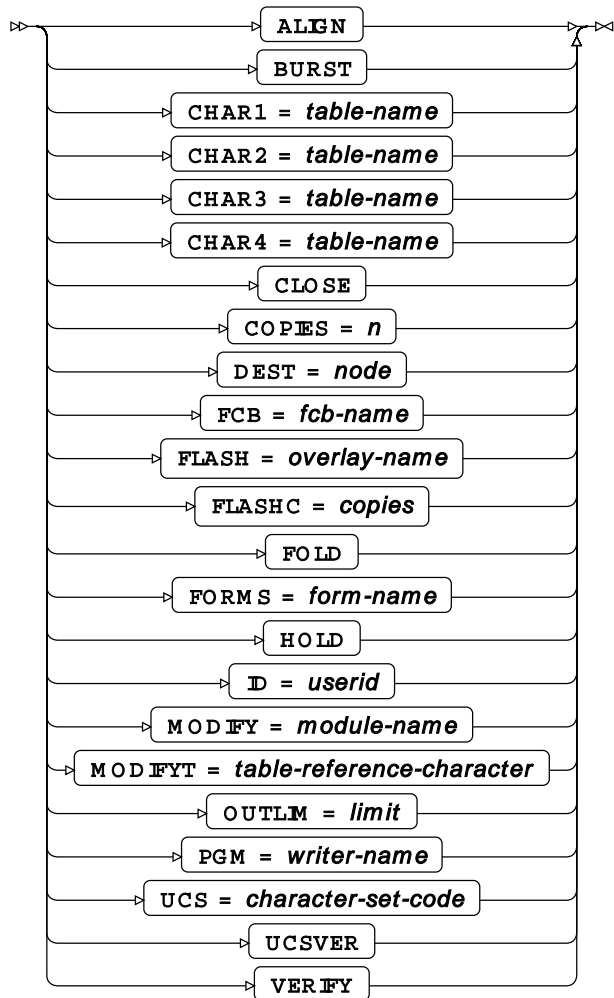
socket-option



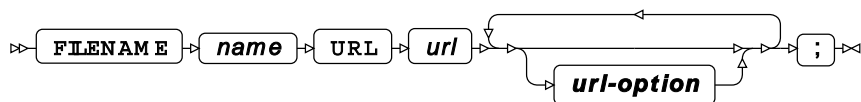
FILENAME, SYSOUT Access Method



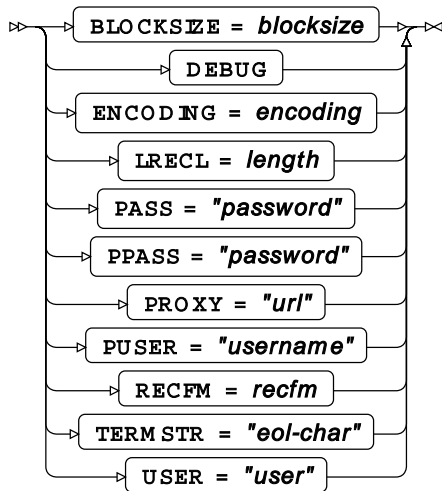
option



FILENAME, URL Access Method



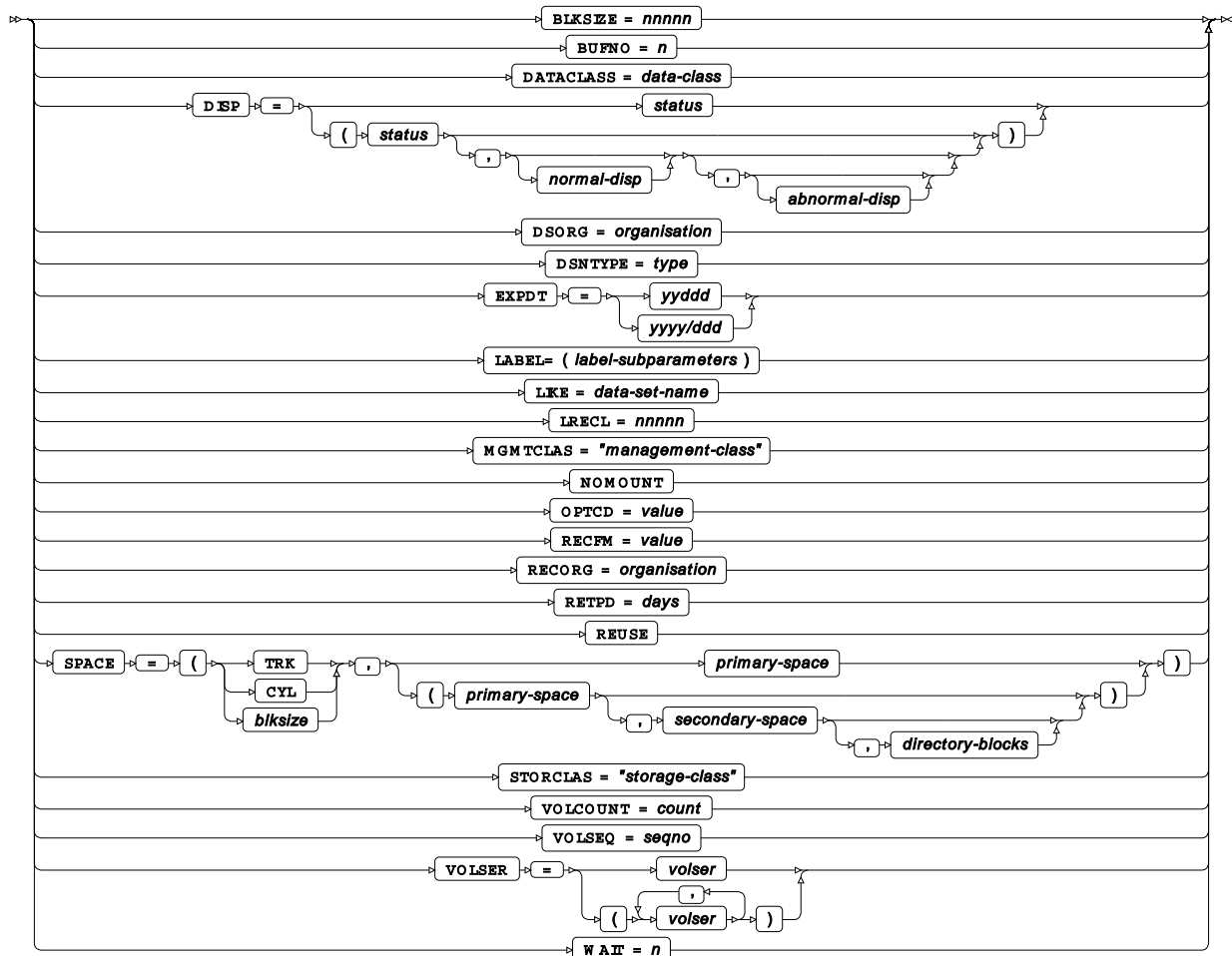
url-option



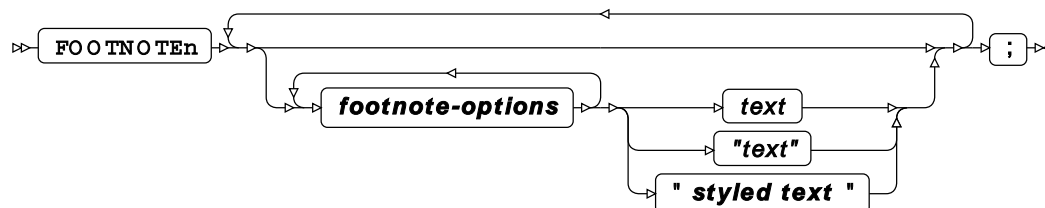
FILENAME, z/OS Datasets Access Method



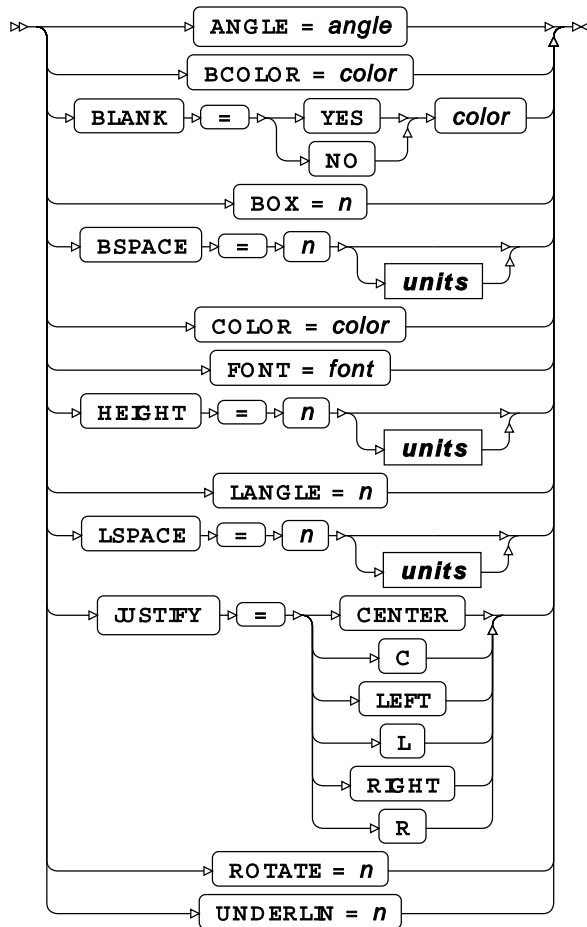
option



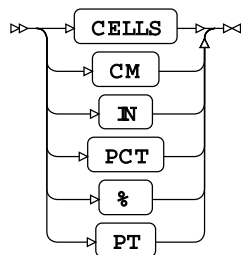
FOOTNOTE



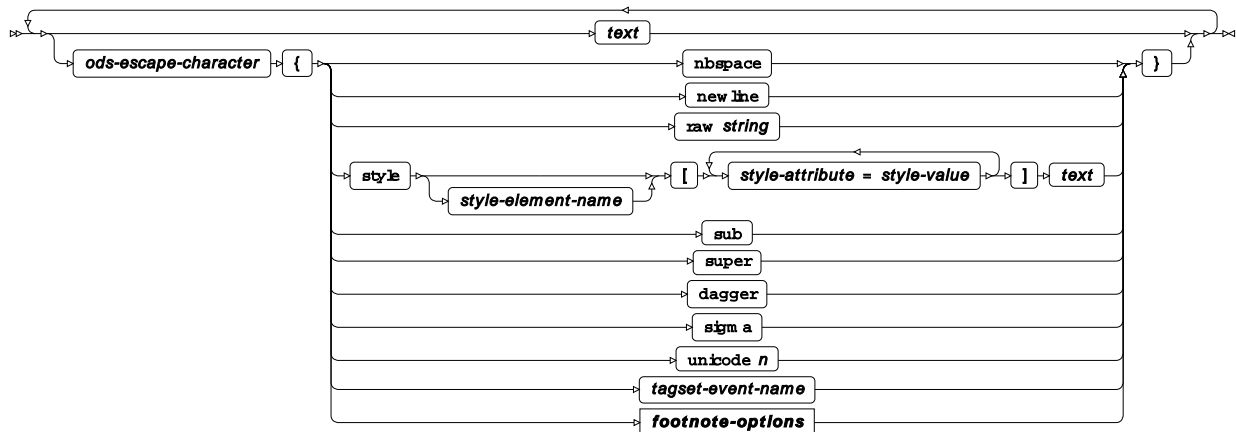
footnote-options



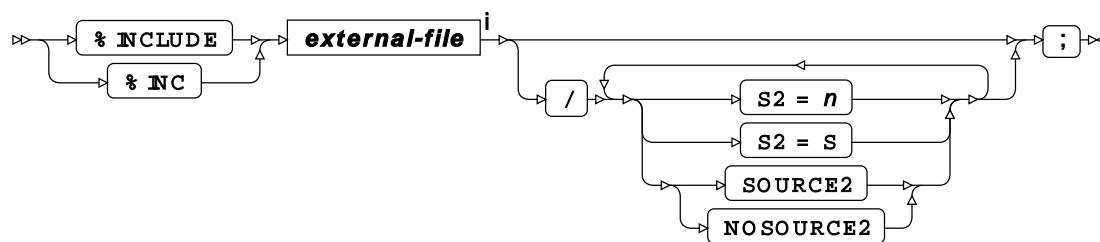
units



styled text

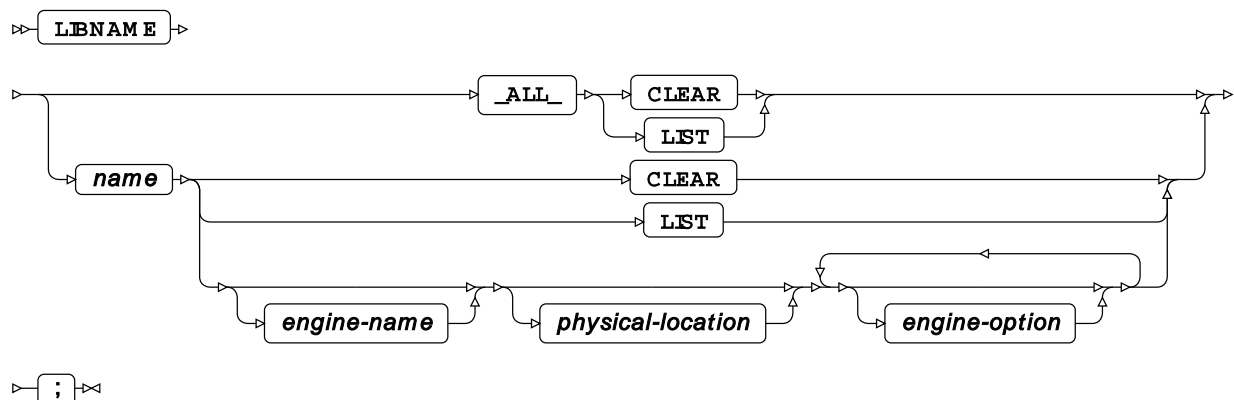


%INCLUDE

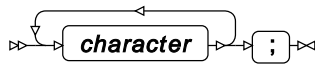


ⁱ See *External Files* [↗](#) (page 29).

LIBNAME

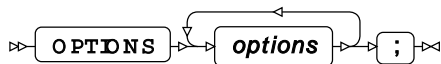


MISSING



OPTIONS

Enables you to set and modify system options.



System options are settings that to the environment in which you run SAS language programs. The system options are described in *System options* [↗](#) (page 42).

Options

options can be one of more of:

APPEND

Append a string to the value of a specified system option.

```
APPEND = ( system-option = string )
```

Use **APPEND** to add information to a system option that has already set. The appended string is separated from any already existing string in the system option by a space. You can only append information to system options that have the `appendable` attribute . See *System options* for more information.

system-option is the system option to be modified. *string* is the string to be appended. This can be any string appropriate to the system option.

INSERT

Insert a string at the start of the value of a specified system option.

```
INSERT = ( system-option = string )
```

Use **INSERT** to add information to a system option that has already been set. The inserted string is separated from any already existing string in the system option by a space. You can only insert information in system options that have the attribute `appendable`. See *System options* for more information.

system-option is the system option to be modified. *string* is the string to be inserted. This can be any string appropriate to the system option.

RESTRICT

Restrict a specified system option.

➤ **RESTRICT** = (*system-option*) ➤

Use **RESTRICT** to restrict changes to system options. A restricted system option cannot be changed during the session in which it is restricted. You can only restrict system options that have the attribute `restrictable`. For information on restricting system options, see *Restricting system options* [↗](#) (page 46). If you only specify one system option, you do not need the parentheses.

system-option

Specify a system option.

➤ *system-option* ➤

The structure of a system option depends on the option. Some system options are switches; for example, `CAPS` and `NOCAPS`. Some system options enable you to set values; for example, `DATESTYLE = DMY`.

For information on the available system options, see *System options* [↗](#) (page 42).

Basic example

In this example, the **OPTIONS** statement is used to specify the amount of memory used to sort data.

```
OPTIONS SORTSIZE = 2G;
LIBNAME books 'c:\temp\books';
PROC SORT DATA=books.lib_books OUT=temp;
BY author title;
```

Example – restricting a system option

In this example, the **OPTIONS** statement is used to set a folder to be searched for macros, and then that system option is restricted so that the location cannot be changed during the rest of the WPS session.

```
OPTIONS SASAUTOS = 'C:\macros' RESTRICT= SASAUTOS;
```

Example – adding information to a system option

In this example, the **INSERT** option of the **OPTIONS** statement is used to add a location to the **SASAUTOS** system option. This example assumes that **SASAUTOS** has been set to `c:\macros` in a configuration file.

```
LIBNAME ms 'c:\temp';
OPTIONS INSERT=(SASAUTOS = ms);
```

This inserts the library reference `ms` before the current location specified in **SASAUTOS**; therefore `ms` will be searched first for autocall macros.

You can use the OPTIONS procedure to display the current setting of the SASAUTOS system option:

```
PROC OPTIONS OPTION = SASAUTOS SHORT
```

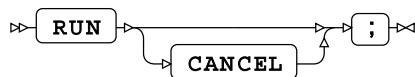
In this example, this would display the following in the log:

```
SASAUTOS=(ms "C:\macros")
```

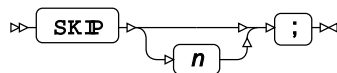
PAGE



RUN

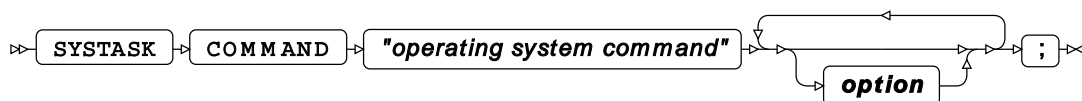


SKIP

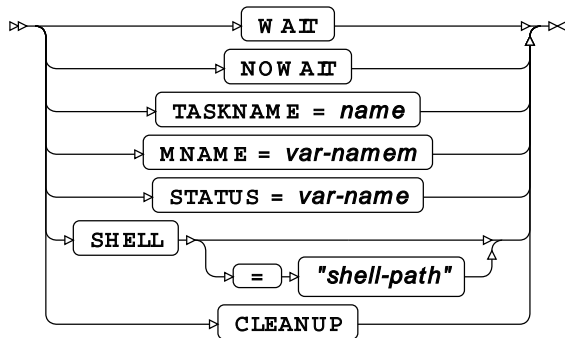


SYSTASK statements

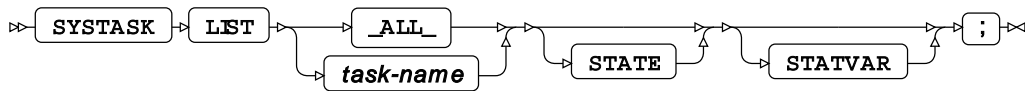
SYSTASK COMMAND



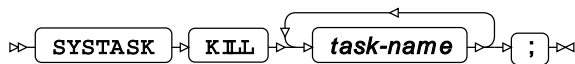
option



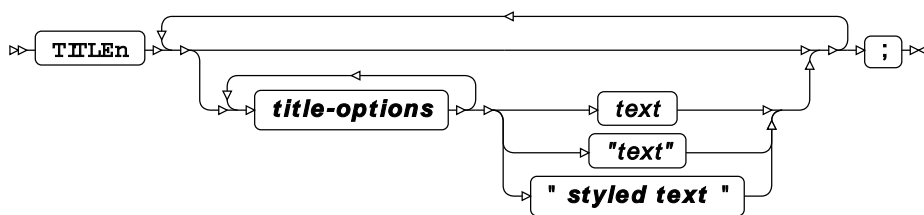
SYSTASK LIST



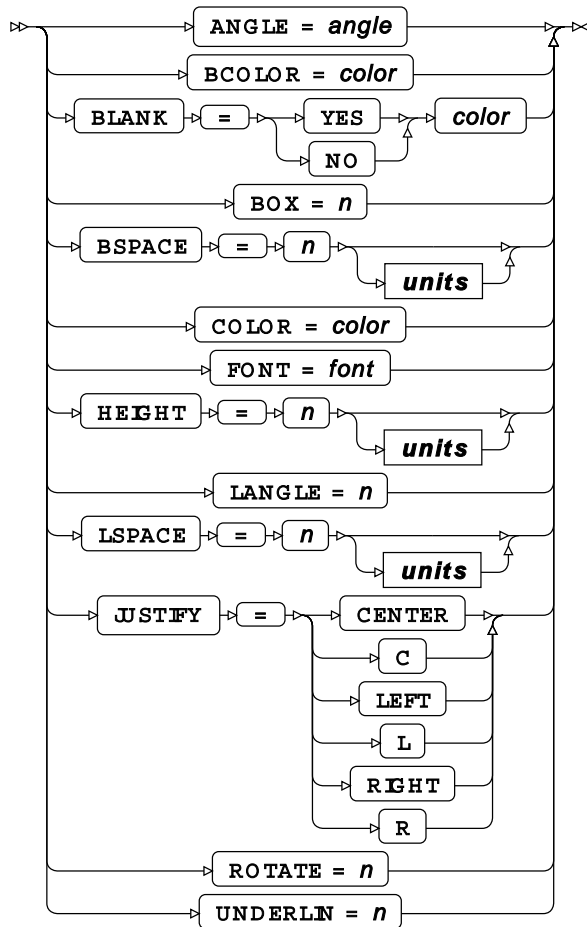
SYSTASK KILL



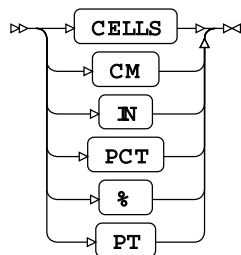
TITLE



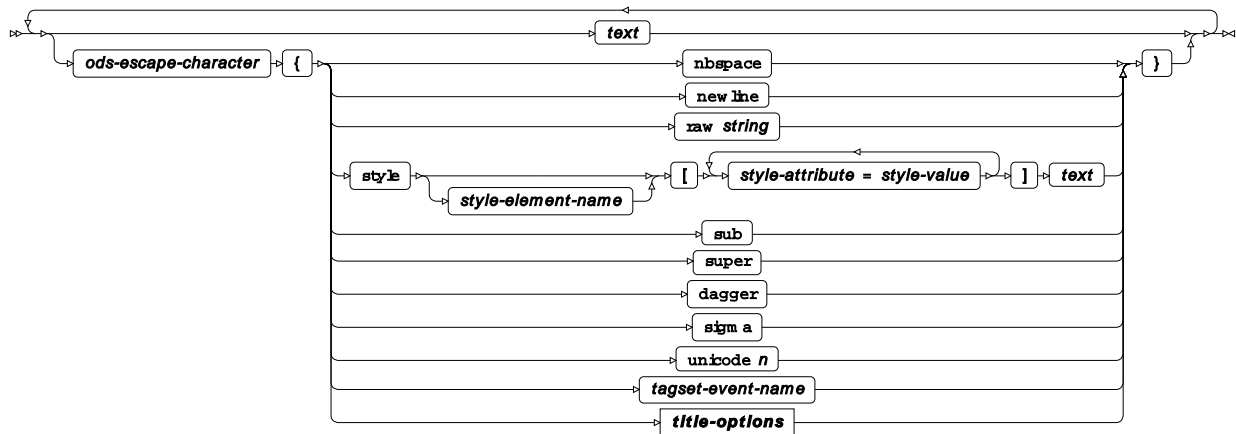
title-options



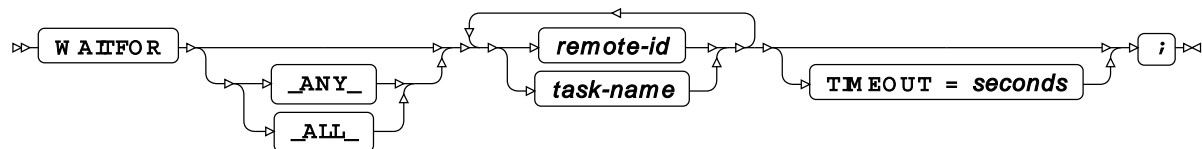
units



styled text



WAITFOR



The `WAITFOR _ALL_` statement suspends execution of the current session until processing is complete for **all** of the `task-names` (or for **all** of the server `remote-ids` in the case of WPS Communicate), or until the `TIMEOUT` interval, if specified, has expired.

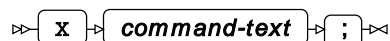
If you use `WAITFOR _ANY_`, or simply `WAITFOR`, instead of `WAITFOR _ALL_`, then execution of the session will only be suspended until processing is complete on one of the server `task-names` or `remote-ids` (or until the `TIMEOUT` interval, if specified, has expired).

Note:

As implied above, the default is `_ANY_` rather than `_ALL_` if no argument is supplied between `WAITFOR` and the `remote-ids` or `task-names`.

X statements

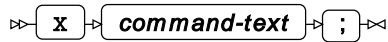
X (on UNIX platforms)



X (on Windows)

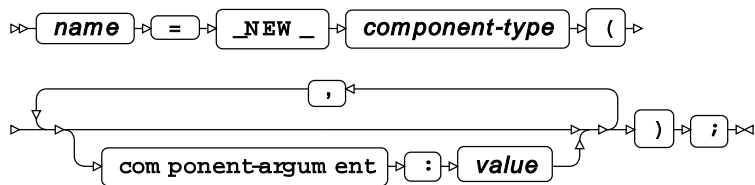


X (on z/OS)

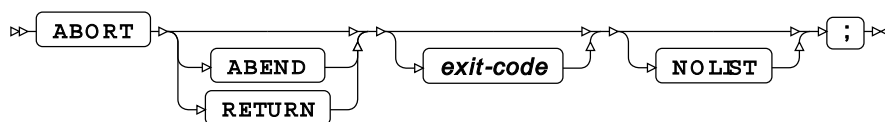


DATA step statements

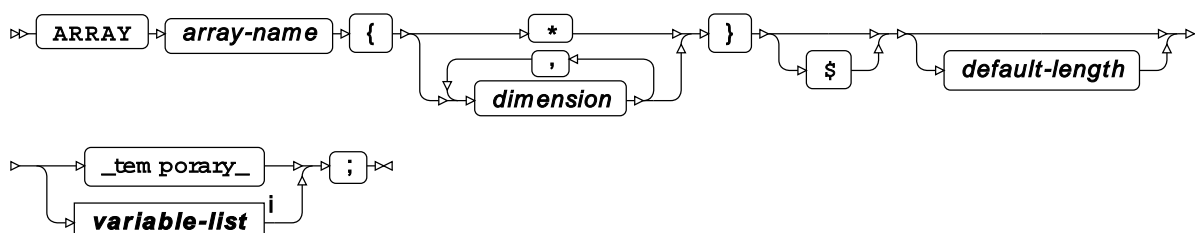
NEW



ABORT

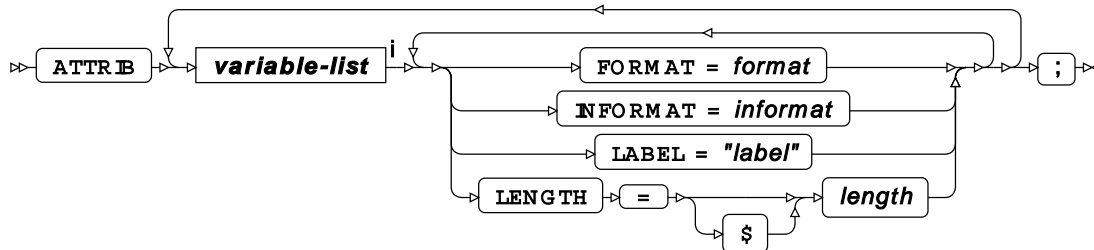


ARRAY



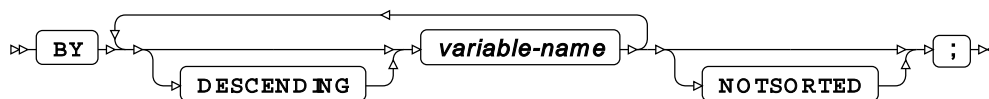
ⁱ See [Variable Lists](#) (page 32).

ATTRIB

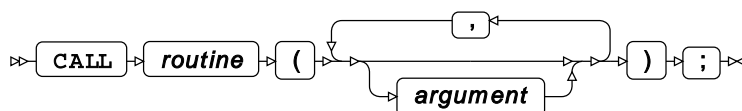


ⁱ See [Variable Lists](#) (page 32).

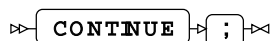
BY



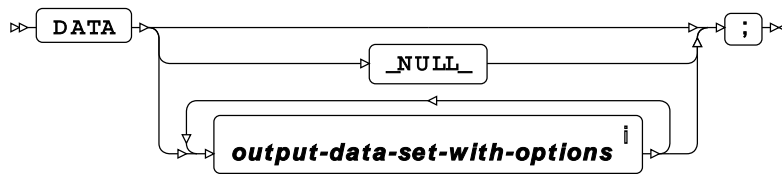
CALL



CONTINUE

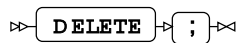


DATA

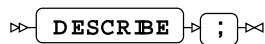


ⁱ See [Output dataset](#) (page 16).

DELETE

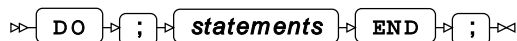


DESCRIBE

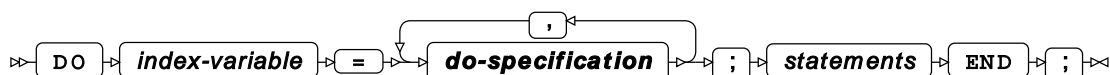


This data step statement is only valid for data step views or stored compiled data steps and instructs WPS to print the source of a stored compiled data step or data step view to the system log. It prevents the implicit execution of a stored compiled data step.

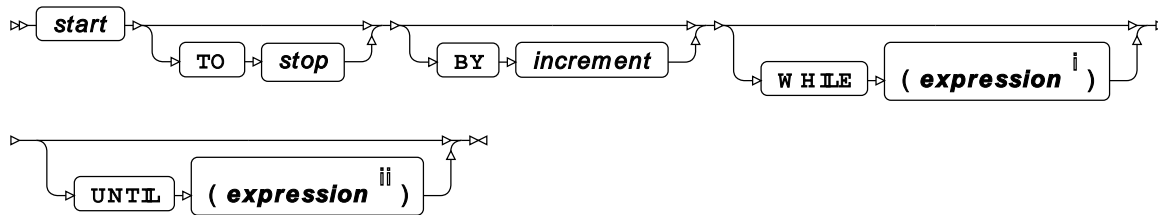
DO



DO, iterative



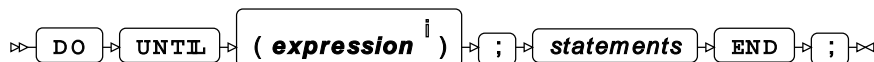
do-specification



ⁱ See *SAS Language expressions* [↗](#) (page 24).

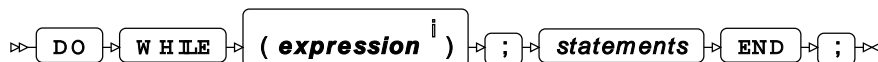
ⁱⁱ See *SAS Language expressions* [↗](#) (page 24).

DO UNTIL



ⁱ See *SAS Language expressions* [↗](#) (page 24).

DO WHILE



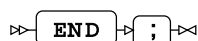
ⁱ See *SAS Language expressions* [↗](#) (page 24).

DROP

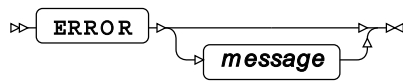


ⁱ See *Variable Lists* [↗](#) (page 32).

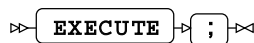
END



ERROR



EXECUTE



This data step statement is only valid for stored compiled data steps and instructs WPS to execute a stored compiled dataset. It is implicit unless the `DESCRIBE` data step statement has been specified.

FILE

In general terms, a **FILE** statement identifies an external file to be used by the DATA step into which to write output from a corresponding [PUT](#) (page 639) statement.

When used with ODS (`FILE PRINT ODS`), it lists the variables to include in the ODS output, and must be used if you have specified the `_ODS_` option in the **PUT** statement.

Note:

The `_ODS_` option determines that values are written to the data component for each of the variables defined as columns via the `COLUMNS` statement below, using the number of lines specified via `N = Number` below.

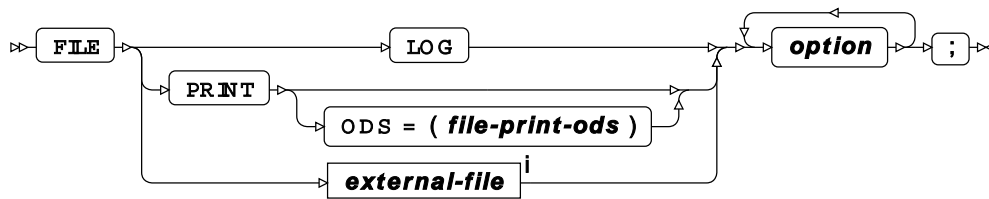
Caution:

The **FILE-PRINT-ODS** statement must precede the **PUT _ODS_** statement in the DATA step.

When the **FILE** statement tries to write beyond the final column, the resultant behaviour can be controlled using the following overflow controls:

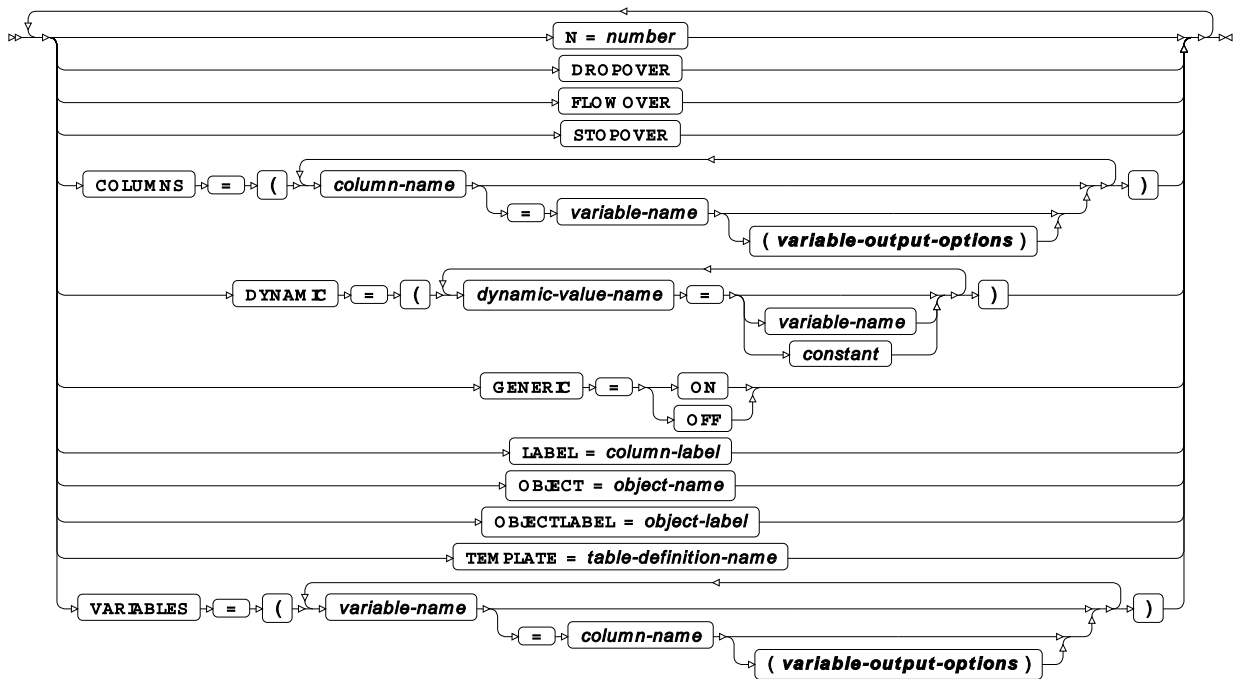
- **DROPOVER.** This discards those values that would otherwise be written beyond the final column.
- **FLOWOVER.** This creates new lines for those values that would otherwise be written beyond the final column.
- **STOPOVER.** This immediately terminates processing of the DATA step, and generates an error message.

Syntax:

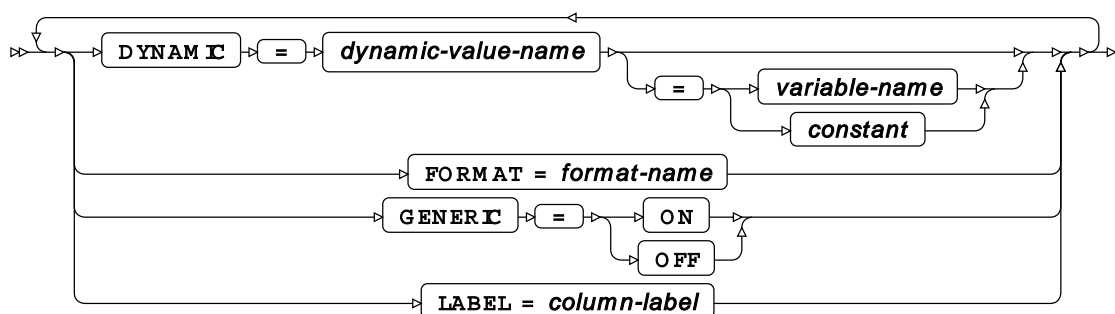


ⁱ See *External Files* [↗](#) (page 29).

file-print-ods



variable-output-options

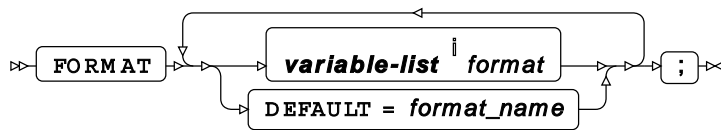


```

graph TD
    BLKSIZE[BLKSIZE] --> BSIZE[block-size]
    BUFND[BUFND] --> BUFND[bufnd]
    BUFNI[BUFNI] --> BUFNI[bufni]
    CLOSE[DISPOSITION] --> DISPOSITION[disposition]
    COLUMN[COL] --> COL[variable]
    DCB[DCB] --> FILEREF[fileref]
    DELIMIT[DELIMITER] --> DELIM[delimiter-char]
    DIM[character-variable] --> DEVTYPE[variable]
    DIMSOFT[DIMSOFT] --> DIMSTR[variable]
    DIMSTR[character-variable] --> DROPOVER[DROPOVER]
    DSCB[DSCB] --> DSD[DSD]
    FEEDBACK[FEEDBACK] --> FBK[variable]
    FLEN[FILENAME] --> FLEVAR[variable]
    FLOW[FLOW OVER] --> FOOTNOTES[FOOTNOTES]
    FOOTNOTES[FOOTNOTE] --> NOFOOTNOTES[NOFOOTNOTES]
    HEADER[HEADER] --> IGNORED[IGNORED]
    JCB[JCB] --> KEYLEN[variable]
    KEYPOS[KEYPOS] --> LNE[variable]
    LINESIZE[LINESIZE] --> LINESLEFT[variable]
    LRECL[LRECL] --> MOD[MOD]
    MOD[available-lines] --> PAGESIZE[PAGESIZE]
    PAGESIZE[PS] --> OLD[OLD]
    OLD[FAD] --> NOPAD[NOPAD]
    PAGESIZE[PAGESIZE] --> PASSWD[PASSWD]
    PASSWD[delimiter-string] --> PRINT[PRINT]
    PRINT[NO PRINT] --> RECFM[RECFM]
    RECFM[D] --> F[F]
    RECFM[FB] --> N[N]
    RECFM[P] --> S[S]
    RECFM[S370V] --> S370VB[S370VB]
    RECFM[S370VS] --> S370VBS[S370VBS]
    RECFM[S370VSTREAM] --> S370VBSSTREAM[S370VBSSTREAM]
    RECFM[U] --> V[V]
    RECFM[VB] --> RECORDS[variable]
    RESET[RESET] --> RRN[variable]
    STOPOVER[STOPOVER] --> TERMSTR[TERMSTR]
    TERMSTR[CR] --> CR[CR]
    TERMSTR[CRLF] --> LF[LF]
    TERMSTR[NL] --> NULL[NULL]
    TITLES[TITLES] --> TITLE[TITLE]
    TITLES[NOTITLES] --> NOTITLE[NOTITLE]
    UCBNAME[UCBNAME] --> VOLUME[variable]
    VOLUME[VOLUMES] --> VSAM[VSAM]
    FILE[FILE] --> FILE[variable]

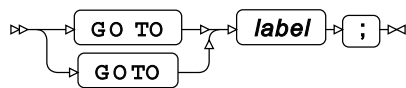
```

FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

GO TO

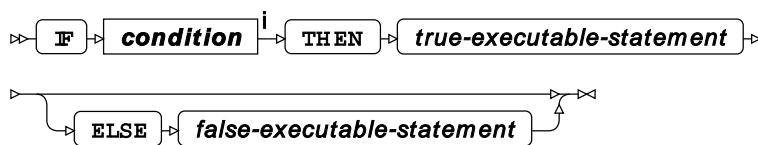


IF, subsetting



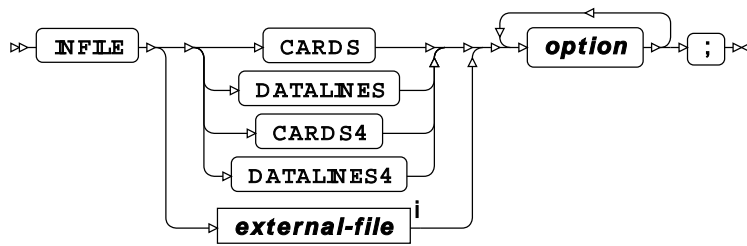
ⁱ See *SAS Language expressions* [↗](#) (page 24).

IF-THEN/ELSE



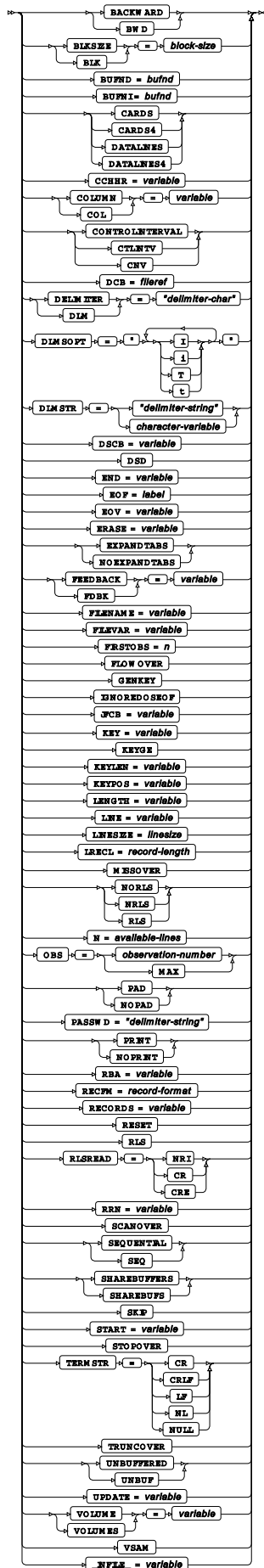
ⁱ See *SAS Language expressions* [↗](#) (page 24).

INFILE

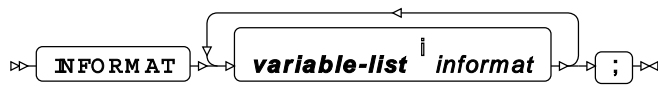


ⁱ See *External Files* [↗](#) (page 29).

option

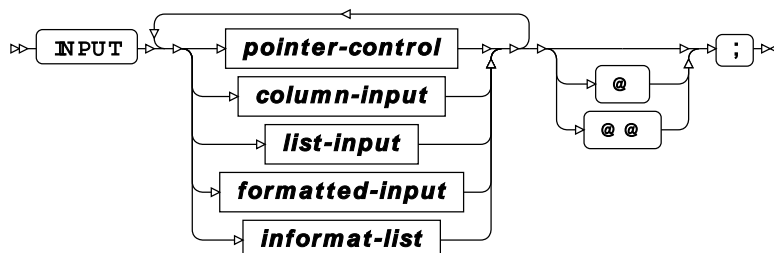


INFORMAT

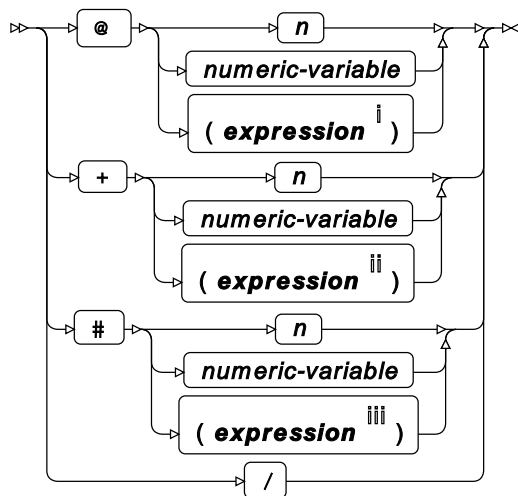


ⁱ See *Variable Lists* [↗](#) (page 32).

INPUT



pointer-control

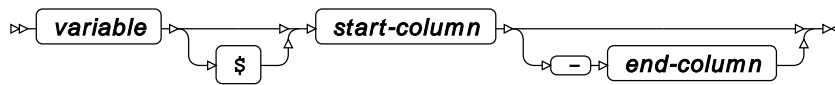


ⁱ See *SAS Language expressions* [↗](#) (page 24).

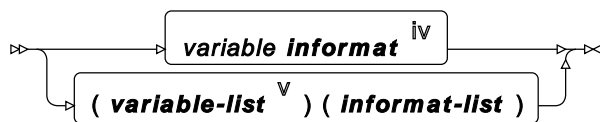
ⁱⁱ See *SAS Language expressions* [↗](#) (page 24).

ⁱⁱⁱ See *SAS Language expressions* [↗](#) (page 24).

column-input



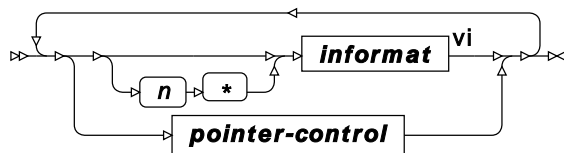
formatted-input



^{iv} See *INFORMAT* [↗](#) (page 635).

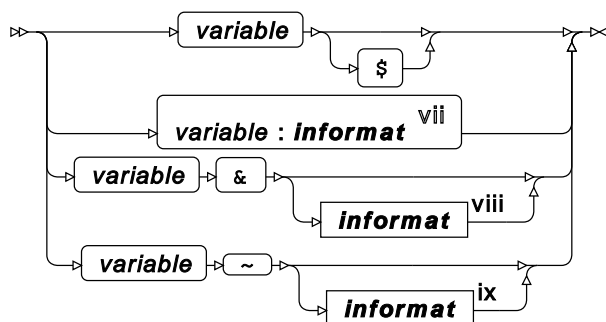
^v See *Variable Lists* [↗](#) (page 32).

informat-list



^{vi} See *INFORMAT* [↗](#) (page 635).

list-input



^{vii} See *INFORMAT* [↗](#) (page 635).

^{viii} See *INFORMAT* [↗](#) (page 635).

^{ix} See *INFORMAT* [↗](#) (page 635).

KEEP



ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



ⁱ See *Variable Lists* [↗](#) (page 32).

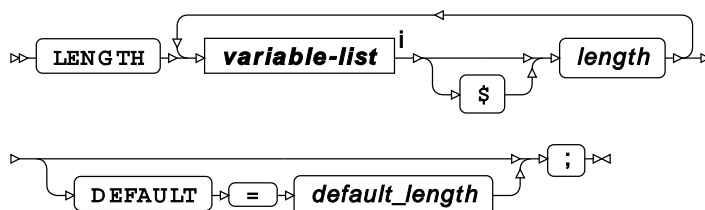
Labels, Statement



LEAVE



LENGTH



ⁱ See *Variable Lists* [↗](#) (page 32).

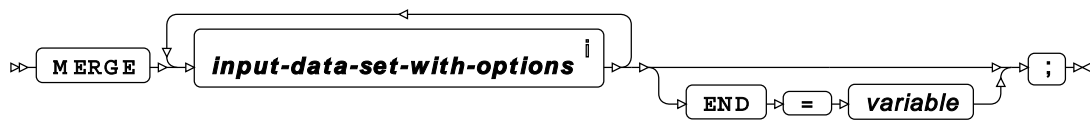
LINK



LIST

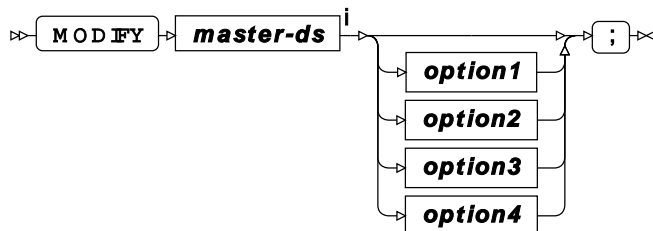


MERGE



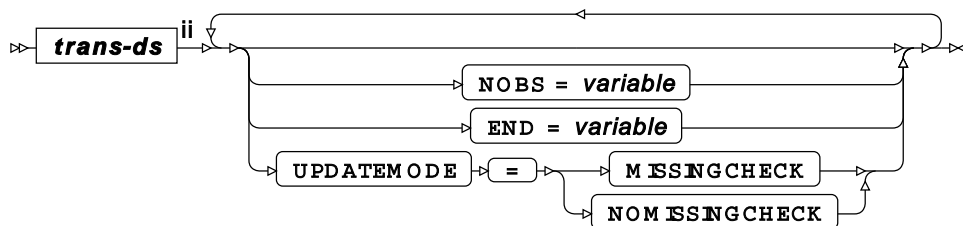
ⁱ See [Input dataset](#) (page 16).

MODIFY



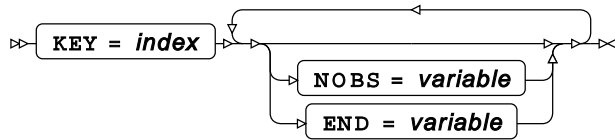
ⁱ See [Input dataset](#) (page 16).

option1

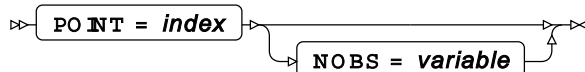


ii See *Input dataset* [↗](#) (page 16).

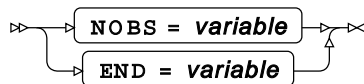
option2



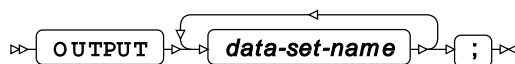
option3



option4



OUTPUT



PUT

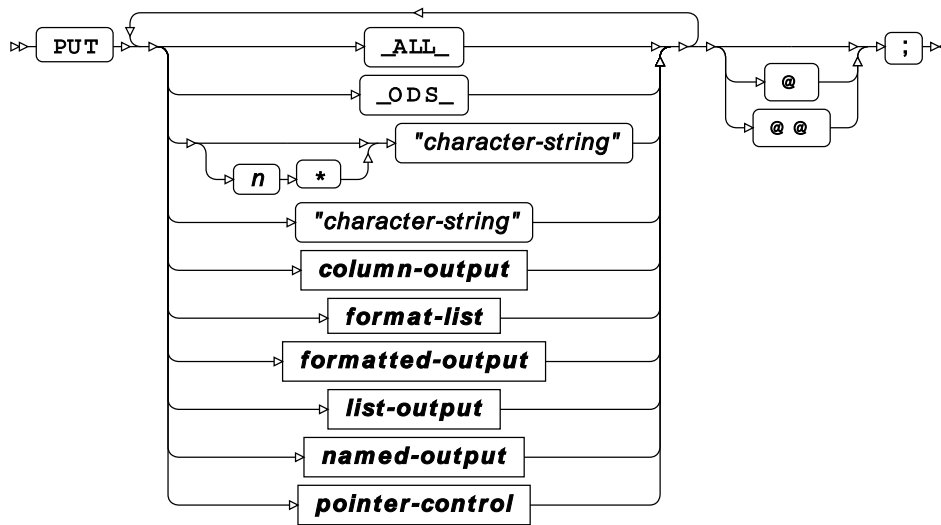
In general terms, a `PUT` statement determines which lines are written, and controls how and where they are written. When output is to be written to an external file, this is specified in a corresponding *FILE* [↗](#) (page 629) statement.

When used with the `_ODS_` option, it writes values to the data component for each of the variables defined as columns via the `COLUMNS` statement in the `FILE-PRINT-ODS` statement (refer to the *FILE* [↗](#) (page 629) statement).

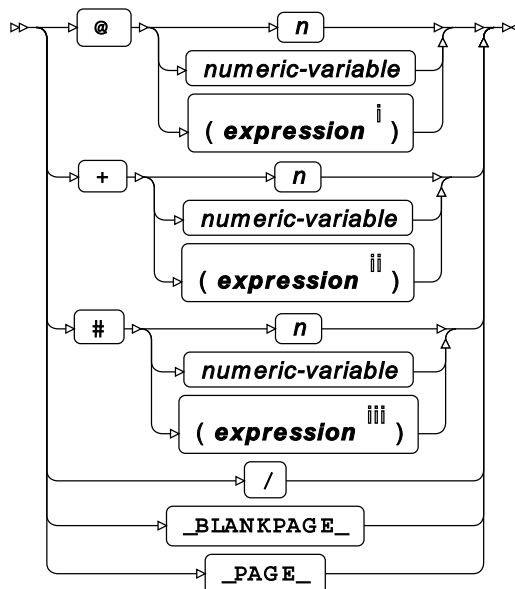
Caution:

The `FILE-PRINT-ODS` statement must precede the `PUT _ODS_` statement in the `DATA` step.

Syntax:



pointer-control

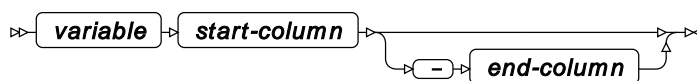


ⁱ See *SAS Language expressions* [↗](#) (page 24).

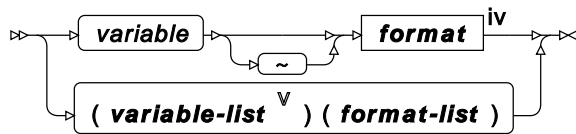
ⁱⁱ See *SAS Language expressions* [↗](#) (page 24).

ⁱⁱⁱ See *SAS Language expressions* [↗](#) (page 24).

column-output



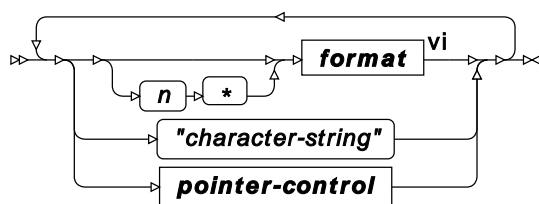
formatted-output



iv See *FORMAT* [↗](#) (page 632).

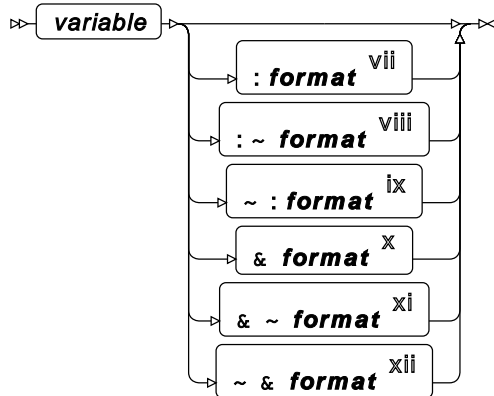
v See *Variable Lists* [↗](#) (page 32).

format-list



vi See *FORMAT* [↗](#) (page 632).

list-output



vii See *FORMAT* [↗](#) (page 632).

viii See *FORMAT* [↗](#) (page 632).

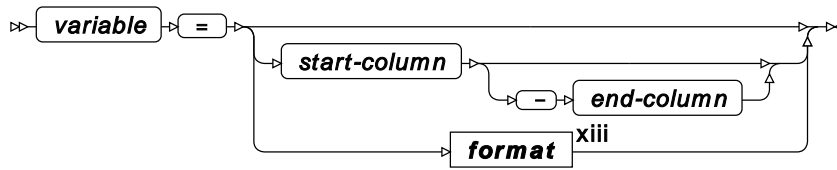
ix See *FORMAT* [↗](#) (page 632).

x See *FORMAT* [↗](#) (page 632).

xi See *FORMAT* [↗](#) (page 632).

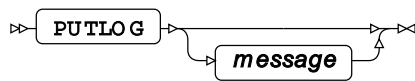
xii See *FORMAT* [↗](#) (page 632).

named-output

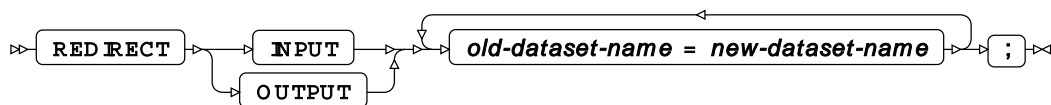


xiii See *FORMAT* [↗](#) (page 632).

PUTLOG

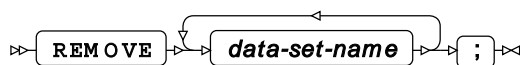


REDIRECT

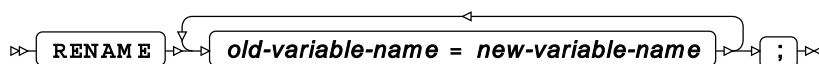


This data step statement is only valid for stored compiled data steps and changes which datasets the stored compiled data step will use for input or output. It may be used multiple times in a data step.

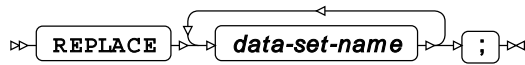
REMOVE



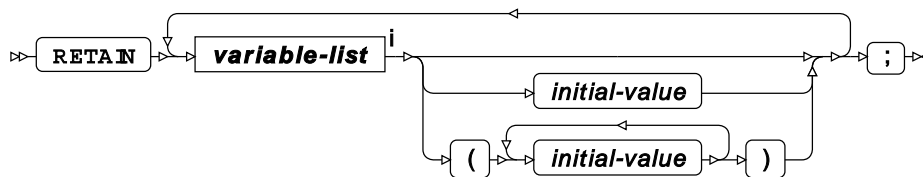
RENAME



REPLACE



RETAIN

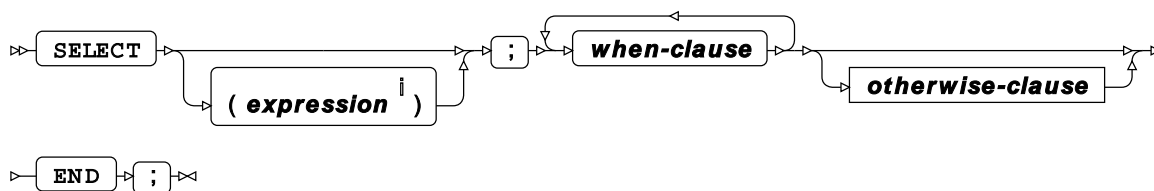


ⁱ See [Variable Lists](#) (page 32).

RETURN

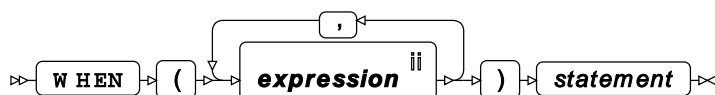


SELECT



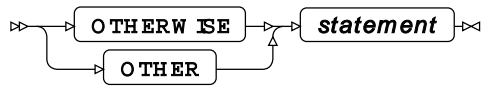
ⁱ See [SAS Language expressions](#) (page 24).

when-clause



ⁱⁱ See [SAS Language expressions](#) (page 24).

otherwise-clause

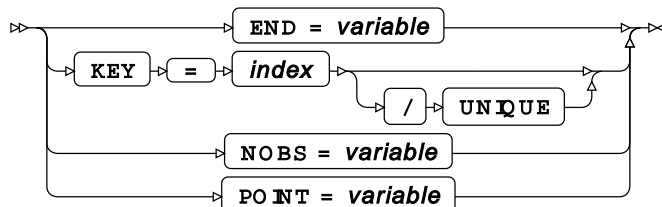


SET

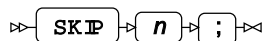


ⁱ See *Input dataset* [↗](#) (page 16).

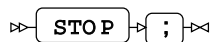
option



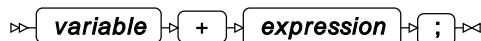
SKIP



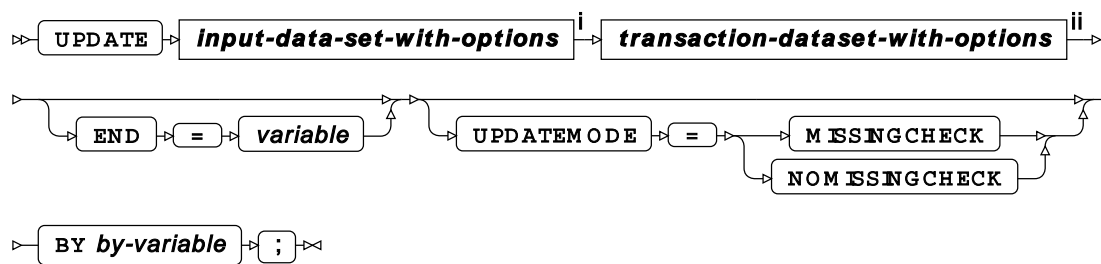
STOP



Sum



UPDATE

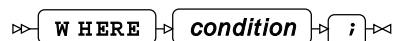


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

WHERE

Restricts the observations to be processed.

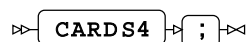


Describing data in a DATA step

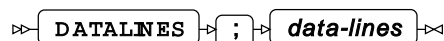
CARDS



CARDS4



DATALINES



DATALINES4

» **DATALINES4** » ; » *data-lines* »

DATA step functions and CALL routines

The **DATA** step functions and **CALL** routines enable you to read data from and write data to a variety of sources, such as datasets and files in formats external to WPS. The data can be manipulated in various ways to make it ready for use by other functions or procedures.

You can, for example, use mathematical and statistical functions to operate on numeric data before presenting the data in graphs or writing it to other files. You can manipulate character-based data to find strings and replace strings, or concatenate data to make other strings. You can find and remove data not required by other operations, such as separators or quotation marks, or you can add separators or quotation marks.

If a **DATA** step function returns a character value to an unformatted specified variable, the variable is created with a default length of 200 characters.

DATA step functions and **CALL** routines available are grouped by operation.

Array functions ↗	648
Return the dimensions and bounds of an array.	
Bitwise functions ↗	652
Manipulate bits in variables using bitwise operations.	
Combination functions and CALL routines ↗	658
Perform combination and permutation operations on specified item.	
Comparison functions ↗	671
Compare strings and numbers and return values based on whether the result is true or false.	
Cryptographic functions ↗	674
Create cryptographic hashes.	
Date and time functions and CALL routine ↗	684
Select, convert and calculate dates and times; for example, you can convert dates to Julian dates, get the current date, count completed time intervals, or return the weekday number.	
Dataset input and output functions and CALL routines ↗	723
Open and close datasets, get information about datasets, and get observations from them.	
Decision forest functions and CALL routines ↗	764
Access decision forest functionality.	
Difference and lag functions ↗	768
Find the difference or lag between two variables.	

Distribution-based functions and CALL routines ↗	775
Perform statistical operations on various probability distributions, including density, survival, quantile and deviance calculations, and drawing random numbers.	
External file functions ↗	1367
Open and close files that have formats external to WPS, and perform other operations using those files.	
External module functions and CALL routines ↗	1420
Execute code stored in external modules.	
Financial functions ↗	1434
Get information about various kinds of financial transactions, such as investments, assets, risk and so on.	
Internet functions ↗	1578
Send and receive information from internet-based resources. These functions can only be used in programs executed by an Application Server with a WPS Web application.	
List functions and CALL routines ↗	1593
Manipulate variables in lists. Lists contain values identified either by their position in the list, or by names.	
ISPF CALL routines ↗	1741
Use the ISPF service to perform tasks.	
Macro functions and CALL routines ↗	1743
Manipulate macro variables and execute macros in the <code>DATA</code> step.	
Mathematical functions and CALL routines ↗	1757
Perform mathematical operations on the data.	
Memory manipulation functions ↗	1880
Manipulate memory.	
Miscellaneous functions ↗	1894
Miscellaneous functions.	
National language support functions ↗	1899
Acquire and set information relating to locales.	
Regular expression functions and CALL routines ↗	1913
Find and manipulate strings using regular expressions.	
Sequence manipulation functions ↗	1932
Operate on sequences (lists) of items in variables.	
String functions and CALL routines ↗	1941
Manipulate strings and characters. Strings are sequences of one or more characters.	
System command function and CALL routine ↗	2110
Execute system commands and run executable files.	
System information functions ↗	2113
Return information about the operating system and WPS.	
Truncation and rounding functions ↗	2120
Define how numbers will be truncated or rounded.	

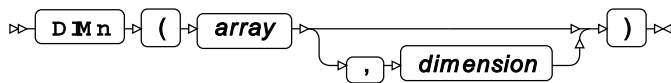
Unicode functions ↗	2146
Convert strings and characters to Unicode format.	
Value formatting and assignment functions ↗	2151
Format and assign data to variables.	
Variable information functions and CALL routines ↗	2163
These functions return information about variables assigned in the DATA step, either explicitly or from a dataset.	
Web functions ↗	2215
Convert the text in URLs and HTML files to different forms.	
Zipcode functions ↗	2221
Accesses information in the ZIPCODE dataset and returns city names, state numbers, state codes, state names, and distances between locations.	

Array functions

Return the dimensions and bounds of an array.	
DIM ↗	648
Returns the size of the specified dimension for an array.	
HBOUND ↗	649
Returns the upper bound of a specified dimension for an array.	
LBOUND ↗	651
Returns the lower bound of a specified dimension for an array.	

DIM

Returns the size of the specified dimension for an array.



You can specify the dimension for which you want information in two ways:

- By using the *dimension* argument, described below.
- By appending a value *n* to DIM.

You can only use one method. If you supply both *n* and *dimension*, an error occurs.

If no dimension is specified, the size of the first dimension is returned.

Return type: Numeric

array**Type:** Array

The name of the array.

dimension

Optional argument

Type: Numeric

The dimension for which the size is returned. By default, this is 1 (the first dimension).

Example

In this example, the function is used to find the number of elements of the specified array dimensions. The result is written to the log.

```
data _null_;

  array a1 {1, 3, 2} a b c d e f;

  rc = dim(a1);
  put "Number of elements: " rc;

  rc = dim(a1,2);
  put "Number of elements: " rc;

  rc = dim3(a1);
  put "Number of elements: " rc;

run;
```

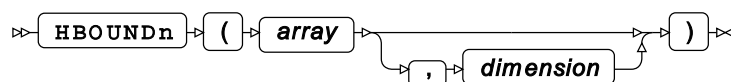
This produces the following output:

```
Number of elements: 1
Number of elements: 3
Number of elements: 2
```

In the first use of `DIM` no dimension is specified, so the number of elements in the first dimension is returned.

HBOUND

Returns the upper bound of a specified dimension for an array.



You can specify the dimension for which you want information in two ways:

- By using the *dimension* argument, described below.

- By appending a value *n* to `HBOUND`.

You can only use one method. If you supply both *n* and *dimension*, an error occurs.

If no dimension is specified, the upper bound of the first dimension is returned.

Return type: Numeric

array

Type: Array

The name of the array for which you want information.

dimension

Optional argument

Type: Numeric

The dimension for which the upper bound is to be returned. By default, this is 1 (the first dimension).

Example

In this example, the function is used to find the upper bounds of the three specified dimensions of an array. The result is written to the log.

```
DATA _NULL_;

  ARRAY a1 {1, 3, 2} a b c d e f;

  ubv = HBOUND(a1);
  PUT "Upper bound of first dimension: " ubv;

  ubv2 = HBOUND(a1,2);
  PUT "Upper bound of second dimension: " ubv2;

  ubv3 = HBOUND3(a1);
  PUT "Upper bound of third dimension: " ubv3;

run;
```

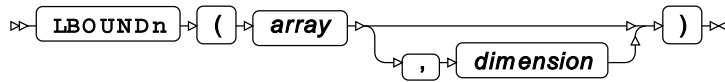
This produces the following output:

```
Upper bound of first dimension: 1
Upper bound of second dimension: 3
Upper bound of third dimension: 2
```

In the first use of `HBOUND` no dimension is specified, so the upper bound of the first dimension is returned.

LBOUND

Returns the lower bound of a specified dimension for an array.



You can specify the dimension for which you want information in two ways:

- By using the *dimension* argument, described below.
- By appending a value *n* to **LBOUND**.

You can only use one method. If you supply both *n* and *dimension*, an error occurs.

If no dimension is specified, the lower bound of the first dimension is returned.

Return type: Numeric

array

Type: Array

The name of the array.

dimension

Optional argument

Type: Numeric

The dimension for which the lower bound is to be returned. By default, this is 1 (the first dimension).

Example

In this example, the function is used to find the lower bounds of the three specified dimensions of an array. The result is written to the log.

```
DATA _NULL_;

  ARRAY a1 {1, 3, 2} a b c d e f;

  lbv = LBOUND(a1);
  PUT "Lower bound of first dimension: " lbv;

  lbv2 = LBOUND(a1,2);
  PUT "Lower bound of second dimension: " lbv2;

  lbv3 = LBOUND3(a1);
  PUT "Lower bound of third dimension: " lbv3;

run;
```

This produces the following output:

```
Lower bound of first dimension: 1
Lower bound of second dimension: 1
Lower bound of third dimension: 1
```

In the first use of `LBOUND` no dimension is specified, so the lower bound of the first dimension is returned.

Bitwise functions

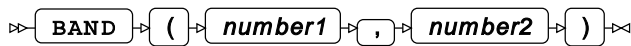
Manipulate bits in variables using bitwise operations.

You might need to use these functions with datasets that contain bit-like data in variables; for example, where *Yes* and *No* are represented by 1 and 0.

<code>BAND</code> ↗	652
Returns the result of combining arguments using a bitwise AND.	
<code>BOR</code> ↗	653
Returns the result of combining two arguments using a bitwise OR.	
<code>BXOR</code> ↗	654
Returns the result of combining two arguments using a bitwise XOR.	
<code>BNOT</code> ↗	655
Returns the result of swapping bit values using a bitwise NOT.	
<code>BLSHIFT</code> ↗	656
Returns the result of shifting the bits of an argument to the left.	
<code>BRSHIFT</code> ↗	657
Returns the result of shifting the bits of an argument to the right.	

BAND

Returns the result of combining arguments using a bitwise AND.



Each corresponding bit in the arguments is compared, and if two corresponding bits are 1, the resulting bit is 1. Any other combination of corresponding bits results in a bit value of 0.

Return type: Numeric

number1

Type: Numeric

The first value to be combined.

number2

Type: Numeric

The second value to be combined.

Each number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, two numbers are combined using the `BAND` function. The result is written to the log.

```
DATA _NULL_;  
    result=BAND(10, 12);  
    PUT "The result is: " result;  
RUN;
```

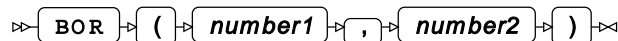
This produces the following output:

```
The result is: 8
```

The numbers are combined as the binaries 1010 and 1100; the function creates the binary 1000, which is returned as the corresponding decimal 8.

BOR

Returns the result of combining two arguments using a bitwise OR.



Combines the values of two arguments using a bitwise OR. Each bit in the arguments is compared; if corresponding bits are 0, the resulting bit is 0. Any other combination of corresponding bits results in a bit value of 1. The result is returned as a numeric value.

Return type: Numeric

number1

Type: Numeric

The first value to be combined.

number2

Type: Numeric

The second value to be combined.

Each number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, two numbers are combined using the `BOR` function. The result is written to the log.

```
DATA _NULL;  
    result = BOR(10, 12);  
    PUT "The result is: " result;  
RUN;
```

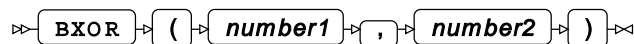
This produces the following output:

```
The result is: 14
```

The numbers are combined as the binaries `1010` and `1100`; the function creates the binary `1110`, which is returned as the corresponding decimal `14`.

BXOR

Returns the result of combining two arguments using a bitwise XOR.



Combines the values of two arguments using a bitwise XOR. Each bit in the arguments is compared; the function adds the bits in the corresponding location of *number1* and *number2*, and discards the carry. The result is 0 when two 0s or two 1s correspond; otherwise the result is 1. For example, if 0 in one argument corresponds with 1 in the other, then the result is 1. The result of the function is returned as a numeric value.

Return type: Numeric

number1

Type: Numeric

The first value to be combined.

number2

Type: Numeric

The second value to be combined.

Each number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, two numbers are combined using the `BXOR` function. The result is written to the log.

```
DATA _NULL_;  
    result = BXOR(10, 12);  
    PUT "The result is: " result;  
RUN;
```

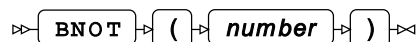
This produces the following output:

```
The result is: 6
```

The numbers are combined as the binaries 1010 and 1100; the function creates the binary 0110, which is returned as the corresponding decimal 6.

BNOT

Returns the result of swapping bit values using a bitwise NOT.



```
>> BNOT ( number ) <<
```

Swaps the values of each bit in an argument using a bitwise NOT. If a bit is 1, then 0 is returned; if a bit is 0, then 1 is returned.

Return type: Numeric

number

Type: Numeric

The value to be bit-swapped.

Each number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, the binary digits of a number are swapped using the `BNOT` function. The result is written to the log.

```
DATA _NULL_;  
    result = BNOT(0);  
    PUT "The result is: " result;  
RUN;
```

This produces the following output:

```
The result is: 4294967295
```

The decimal number 0 is converted to the 32-bit binary number 00000000000000000000000000000000; this function creates the binary 11111111111111111111111111111111 which is returned as the corresponding decimal integer, 4294967295.

BLSHIFT

Returns the result of shifting the bits of an argument to the left.

➤ **BLSHIFT** ➤ (➤ *number* ➤ , ➤ *shift-amt* ➤) ➤

Performs a left bit-shift, which moves the bits in a value to the left. The bits lost by the shift at the left-hand end of the value are replaced by zeros at the right-hand end of the value. You can specify the number of bits to be shifted. For example, if the binary value 0101 is bit-shifted to the left by one position, the result will be 1010. If the binary value 001101 is bit-shifted to the left by two positions, the result will be 110100.

Return type: Numeric

number

Type: Numeric

The value to be bit-shifted to the left.

shift-amt

Type: Numeric

The number of bits to be shifted.

number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, the binary digits of a number are bit-shifted to the left by two digits. The result is written to the log.

```
DATA _NULL_;  
  result = BLSHIFT(75,2);  
  PUT "The result is: " result;  
RUN;
```

This produces the following output:

```
The result is: 300
```

The number 75 corresponds to the binary 001001011; a left-shift of two digits results in 100101100, which is returned as the corresponding decimal, 300.

BRSHIFT

Returns the result of shifting the bits of an argument to the right.

➤ **BRSHIFT** ➤ (➤ *number* ➤ , ➤ *shift-amt* ➤) ➤

Performs a right bit-shift, which moves the bits in a value to the right. The bits lost by the shift at the right-hand end of the value are replaced by zeroes at the left-hand end of the string. You can specify the number of bits to be shifted. For example, if the binary value 101 is bit-shifted to the right by one position, the result will be 010. If the binary value 1101 is bit-shifted to the right by two positions, the result will be 0011.

Return type: Numeric

number

Type: Numeric

The value to be bit-shifted to the right.

shift-amt

Type: Numeric

The number of bits to be shifted.

number is interpreted as a 32-bit integer. If you pass a number that is not an integer, any fraction is ignored and the number converted to an integer. A runtime error is returned if the integer is less than zero or greater than $2^{32}-1$.

Example

In this example, the binary digits of a number are bit-shifted to the right by two digits. The result is written to the log.

```
DATA _NULL_;  
    result = BRSHIFT(75,2);  
    PUT "The result is: " result;  
RUN;
```

This produces the following output:

```
The result is: 18
```

The decimal number 75 corresponds to the binary 001001011; a right-shift of two digits results in 000010010, which is returned as the corresponding decimal 18.

Combination functions and CALL routines

Perform combination and permutation operations on specified item.

Both combinations and permutations are selections of a number of items from a larger number of items; for example, four numbers from ten numbers, or three toys from ten toys. With a combination, order is not important ; the numbers 5, 9, 10 and the numbers 10,5, 9 are the same combination. With a permutation, order matters; the numbers 5, 9, 10 are a different permutation to the numbers 10, 5, 9.

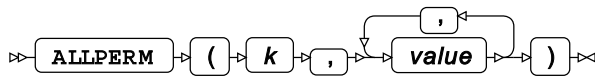
The number of permutations or combinations is also affected by whether repetitions are allowed; that is, whether choosing an item reduces the total number of items from which selection can be made. For example, if you choose three numbers from ten, each choice reduces by one the group of numbers from which you can choose; you can then only create a permutation without repetition. However, if you were to replace each drawn number with the same number, you would be able to create a permutation with repetition.

In these functions, repetition is *not* allowed, for either combinations or permutations.

ALLPERM ↗	659
Returns the ordinal position in a list of permuted items at which the permutation changed from the previous permutation.	
COMB ↗	661
Returns the number of combinations for a specified number of items in a group of items.	
LCOMB ↗	662
Returns the number of combinations for a specified number of items in a group of items and returns the result as a natural logarithm.	
LPERM ↗	663
Returns as a natural logarithm the number of permutations for a specified number of items in a group of items.	
PERM ↗	664
Returns the number of permutations of a group of items.	
CALL ALLPERM ↗	664
Returns the permutation at a specified place in a list of permuted sequences.	
CALL RANPERK ↗	666
Returns a random permutation of the argument list.	
CALL RANPERM ↗	669
Returns a random permutation of the argument list.	

ALLPERM

Returns the ordinal position in a list of permuted items at which the permutation changed from the previous permutation.



The position returned is that of the left-hand item of the pair of items that had to change in the previous permutation to create the specified permutation. Items are permuted beginning from the order of the group of items you specify. The items must be provided as a list of arguments, or as an array.

Return type: Numeric

k

Type: Numeric

The rank of the permutation. The permutations are ranked from 1 to the number of permutations. No repetitions are allowed. For example, the number of permutations of three items with no repetitions allowed is six; the rank of the initial list of items to be permuted is 1, the rank of the final permutation is numbered 6.

value

Type: Var

An item to be permuted.

value must be an item stored in an argument, or an array. Each item must have the same number of characters; if they do not, use formatting or other functions to make them the same.

Basic example

In this example, the function returns the position of the left hand item of the pair that changed to create the specified permutation. The result is written to the log.

```
DATA _NULL_;
  var1 = '100';
  var2 = 'egg';
  var3 = 'red';
  result1 = ALLPERM(2,var1,var2,var3);
  PUT "The position of the left hand item of the pair that changed is: " result1;
RUN;
```

This produces the following output:

```
The position of the left hand item of the pair that changed is: 2
```

In this example, the permutations (remembering that the function permutes with no repetitions) are:

```
100 egg red
100 red egg
egg 100 red
egg red 100
red 100 egg
red egg 100
```

In the specified line, line two, the items that were changed to make that permutation were `egg` and `red` in line 1, which became `red` and `egg` in line two. The left-hand item of the pair that changed is, therefore, the second item.

Example - listing position of change for all permutations

In this example, the DATA step used in the section *Basic Example* is modified so that all permutations are output. The result is written to the log.

```
DATA _NULL_;
  var1 = '100';
  var2 = 'egg';
  var3 = 'red';
  DO i = 1 TO perm(3);
    result1 = ALLPERM(i,var1,var2,var3);
    PUT result1;
  END;
RUN;
```

All permutations are checked in the DO loop, and the following results are returned and written to the log with the PUT statement:

```
0
2
1
2
1
2
```

The first result is 0; it is the first permutation, the items are as ordered in `var1`, `var2`, `var3` and no changes have been made. The second result is 2, as described in *Basic Example*. The third result is 1; and so on. How the first three results are determined is described in the table below.

100 egg red	First line - no changes have been made, so 0 is returned
100 red egg	<code>red</code> and <code>egg</code> swapped. The first (left-hand) index position of the pair of items that were swapped to make this change is at index position 2.
egg 100 red	<code>100</code> and <code>red</code> moved. The first (left-hand) index position of the pair of items that were moved to make this change is at index position 1.

This pattern continues for each permutation.

COMB

Returns the number of combinations for a specified number of items in a group of items.

➤ **COMB** ➤ (➤ *n* ➤ , ➤ *r* ➤) ➤ ➤

Items are permuted without repetition. The number of combinations returned is also equal to the binomial coefficient for specified values.

Return type: Numeric

n

Type: Numeric

The total number of items in a group.

r

Type: Numeric

The number of items to be combined.

Example

In this example, the function is used to calculate the number of combinations of three items in a group of five items, and then to calculate the number of combinations of four items in a group of ten items. The result is written to the log.

```
DATA _NULL_;

    result=COMB(5,3);
    PUT "The number of combinations is: " result;

    result=COMB(10,4);
    PUT "The number of combinations is: " result;

RUN;
```

This produces the following output:

```
The number of combinations is: 10
The number of combinations is: 210
```

LCOMB

Returns the number of combinations for a specified number of items in a group of items and returns the result as a natural logarithm.

➤ **LCOMB** ➤ (➤ *n* ➤ , ➤ *r* ➤) ➤ ➤

Combinations are made without repetition.

Return type: Numeric

n

Type: Numeric

The total number of items in a group.

r

Type: Numeric

The number of items to be combined.

Example

In this example, the function calculates the number of combinations of three items in a group of five items, and four items in a group of ten items. The result is written to the log.

```
DATA _NULL_;

    result=LCOMB(5,3);
    PUT "The number of combinations is: " result;

    result=LCOMB(10,4);
    PUT "The number of combinations is: " result;

RUN;
```

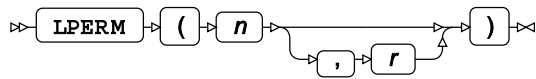
This produces the following output:

```
The number of combinations is: 2.302585093
The number of combinations is: 5.3471075307
```

The first result is the natural logarithm of 10; the second is the natural logarithm of 210.

LPERM

Returns as a natural logarithm the number of permutations for a specified number of items in a group of items.



Return type: Numeric

n

Type: Numeric

The number of items in the group.

r

Optional argument

Type: Numeric

The number of items to be permuted.

Example

In this example, the function calculates the number of permutations of three items in a group of ten items, of three items in a group of five items, and four items in a group of ten items. Repetition is not allowed. The result is written to the log.

```
DATA _NULL_;

    result=LPERM(5,3);
    PUT "The number of permutations is is: " result;

    result=LPERM(10,4);
    PUT "The number of permutations is: " result;

RUN;
```

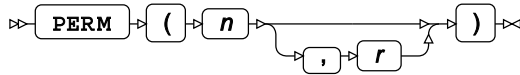
This produces the following output:

```
The number of permutations is: 4.0943445622
The number of permutations is: 8.5251613611
```

The first result is the natural logarithm of 60; the second is the natural logarithm of 5040.

PERM

Returns the number of permutations of a group of items.



The items are permuted without repetition.

Return type: Numeric

n

Type: Numeric

The total number of items in a collection.

r

Optional argument

Type: Numeric

The number of items to be permuted.

Example

In this example, the function calculates the number of permutations of three items in a group of five items, and of four items in a group of ten items. The result is written to the log.

```

DATA _NULL_;

    result=PERM(5,3);
    PUT "The number of permutations is: " result;

    result=PERM(10,4);
    PUT "The number of permutations is: " result;

RUN;
  
```

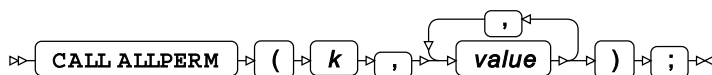
This produces the following output:

```

The number of permutations is: 60
The number of permutations is: 5040
  
```

CALL ALLPERM

Returns the permutation at a specified place in a list of permuted sequences.



Items are permuted beginning from the series of items you specify. The items to be permuted must be provided as a list of variables, or as an array.

The permuted items are returned to the specified variables or array, and replace the original contents.

k

Type: Numeric

The rank of the permutation. The permutations are ranked from 1 to the number of permutations. No repetitions are allowed. For example, the number of permutations of three items with no repetitions allowed is six; the rank of the initial list of items to be permuted is 1, the rank of the final permutation is numbered 6.

value

Type: Var

An item to be permuted.

value must be an item in a variable, or an array.

The value of each item in a variable or array must be the same length; if the lengths vary, use formatting or other functions to make them the same.

Basic example

In this example, the routine writes the specified permutation from the list of all permutations. The result is written to the log.

```
DATA _NULL_;  
  var1=1;  
  var2=2;  
  var3=3;  
  CALL ALLPERM(4, var1,var2,var3);  
  PUT "The fourth permutation is: " var1 var2 var3;  
RUN;
```

This produces the following output:

```
The fourth permutation is: 1 3 2;
```

Example – writing all permutations of the initial items

In this example, the routine creates all permutations of the initial items 1,2 and 3. The result is written to the log.

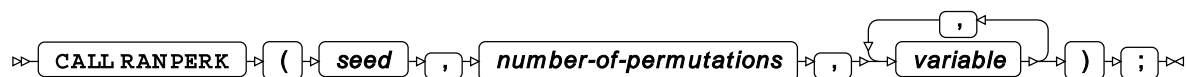
```
DATA _NULL_;
  var1=1;
  var2=2;
  var3=3;
  DO i = 1 TO fact(3);
    CALL ALLPERM(i, var1,var2,var3);
    PUT 'Permutation ' i 'is: ' var1 var2 var3;
  END;
RUN;
```

This produces the following output:

```
Permutation 1 is: 1 2 3
Permutation 2 is: 1 3 2
Permutation 3 is: 3 1 2
Permutation 4 is: 3 2 1
Permutation 5 is: 2 3 1
Permutation 6 is: 2 1 3
```

CALL RANPERK

Returns a random permutation of the argument list.



Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a **DATA** step, the random stream is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same permutation each time the **DATA** step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible permutations from the same **DATA** step. To generate a different permutation each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

Each time you execute this routine, a new random permutation is generated; this includes each use with an updated *seed*. The permutation is generated immediately after the stream has been initialised.

Any items not permuted are returned in the remaining variables or array entries. For example, if you have five items, 1, 2, 3, 4 and 5, and *number-of-permutations* is set to 3, then three of the five numbers are permuted together – say 1, 3, 5 – and returned in the first three variables or array entries; the two numbers not permuted – in this case 2 and 4 – are returned in the final two variables or array entries.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

number-of-permutations

Type: Numeric

The number of permutations.

variable

Type: Var

An array containing the items to be permuted, or a list of variables containing items to be permuted. The items must be of the same type. The permuted items are returned to these variables, or to the array. If strings are used they must be the same length, or defined as the same length.

Examples

In this example, a random permutation is returned on each iteration of the loop. The result of each permutation is returned to the array, and the array is then written to the log.

```
DATA _NULL_;  
  ARRAY rp rp1-rp5 (2 9 100 2 4);  
  DO i = 1 TO 5;  
    CALL RANPERK (2, 4, of rp1-rp5);  
    PUT rp1-rp5;  
  END;  
RUN;
```

This produces the following output:

```
9 4 2 100 2  
2 9 2 100 4  
2 9 100 2 4  
2 9 4 100 2  
4 2 2 100 9
```

In this example, a random permutation is returned on each iteration of the loop. The result of each permutation is returned to the variables, and these variables then written to the log.

```
DATA _NULL_;
  var1=1;
  var2=2;
  var3=3;
  var4=100;
  var5=102;
  DO i = 1 TO 5;
    CALL RANPERK (13,4,var1,var2,var3,var4,var5);
    PUT var1 var2 var3 var4 var5;
  END;
RUN;
```

This produces the following output:

```
3 100 1 2 102
102 2 1 3 100
3 100 102 2 1
3 1 2 100 102
2 102 1 3 100
```

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;
  ARRAY rp rp1-rp5 (2 9 100 2 4);
  DO i = 1 TO 5;
    CALL RANPERK (0, 4, of rp1-rp5);
    PUT rp1-rp5;
  END;
RUN;
```

This produces the following output:

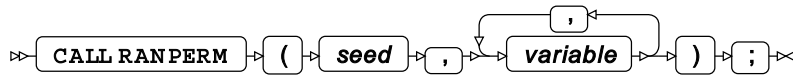
```
9 2 4 2 100
4 9 100 2 2
2 100 9 4 2
9 4 2 2 100
100 9 2 4 2
```

Running the DATA step again produces the following output.

```
2 100 2 4 9
2 4 100 2 9
100 9 2 2 4
2 100 4 9 2
4 2 2 9 100
```

CALL RANPERM

Returns a random permutation of the argument list.

**Important:**

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a **DATA** step, the random stream is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same permutation each time the **DATA** step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible permutations from the same **DATA** step. To generate a different permutation each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

Each time you execute this routine, a new random permutation is generated; this includes each use with an updated *seed*. The permutation is generated immediately after the stream has been initialised.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

variable

Type: Var

An array containing the items to be permuted, or a list of variables containing items to be permuted. The items must be of the same type. The permuted items are returned to these variables, or to the array. If strings are used they must be the same length, or defined as the same length.

Examples

In this example, a random permutation is returned on each iteration of the loop. The result is returned to the array, and then this array is written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  ARRAY rp rp1-rp5 (2 9 100 2 4);  
  DO i = 1 TO 5;  
    CALL RANPERM(2, of rp1-rp5);  
    PUT rp1-rp5;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
2 4 100 2 9  
4 100 2 2 9  
2 2 100 9 4  
4 2 100 2 9  
2 4 9 2 100
```

In this example, a random permutation is returned on each iteration of the loop. These permutations are then written to the log.

```
DATA _NULL_;  
  var1=1;  
  var2=2;  
  var3=3;  
  DO i = 1 TO 5;  
    CALL RANPERM (5,var1,var2,var3);  
    PUT var1 " " var2 " " var3;  
  END;  
RUN;
```

This produces the following output:

```
1 2 3  
2 1 3  
2 3 1  
2 3 1  
3 1 2
```

If the seed is set to 0, each run produces a different series of permutations. For example:

```
DATA _NULL_;  
  ARRAY rp rp1-rp5 (2 9 100 2 4);  
  DO i = 1 TO 5;  
    CALL RANPERM (0, of rp1-rp5);  
    PUT rp1-rp5;  
  END;  
RUN;
```

This produces the following output:

```
4 100 2 2 9
2 100 9 4 2
9 4 2 2 100
2 4 100 2 9
2 9 4 2 100
```

Running the DATA step again produces the following output.

```
100 2 9 4 2
2 9 100 4 2
2 9 4 2 100
100 9 2 2 4
2 100 4 2 9
```

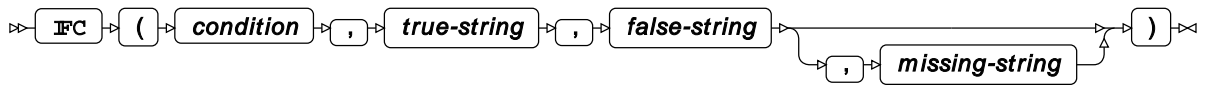
Comparison functions

Compare strings and numbers and return values based on whether the result is true or false.

IFC ↗	671
Returns a string as the result of evaluating a condition. A string is defined for the true and for the false evaluation. Optionally, if the condition is missing, another specified string can be returned.	
IFN ↗	673
Returns a specified result based on a control condition.	

IFC

Returns a string as the result of evaluating a condition. A string is defined for the true and for the false evaluation. Optionally, if the condition is missing, another specified string can be returned.



Return type: Character

condition

Type: Numeric

A valid condition. This can be another function, or any expression.

true-string

Type: Character

The string to be returned if *condition* is true.

false-string

Type: Character

The string to be returned if *condition* is false.

missing-string

Optional argument

Type: Character

The string to be returned if *condition* is missing.

Basic example

In this example, the function returns the text corresponding to the truth value of the first argument. The result is written to the log.

```
DATA _NULL_;  
    result = IFC(1, 'tree', 'flash');  
    PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: tree
```

condition is explicitly set to 1 and is therefore true, so the function returns the string in the *true-string* argument.

Example – condition returned by another function

In this example, the value of *condition* is set by another DATA step function. The result is written to the log.

```
DATA _NULL_;  
    search = 'London';  
    result = IFC(contains('SBC Ltd 2 50 London', search), 'Contains London', 'No  
    London');  
    PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: Contains London
```

In this example, *condition* is set to true (1) by the `CONTAINS` function, which returns 1 if a string contains a specified substring, 0 if not. In this case, the string searched for is `London`; the string is found, so `CONTAINS` returns 1. Because *condition* is true, the function returns the string in the *true-string* argument.

Example – missing value

In this example, the function returns the text specified for the *missing-string*. The result is written to the log.

```
DATA _NULL_;
  result = IFC(choosen(3, 1, 100, var1, 7, 12), 'Big Number', 'Small Number',
  'Missing');
  PUT "The string returned is: " result;
RUN;
```

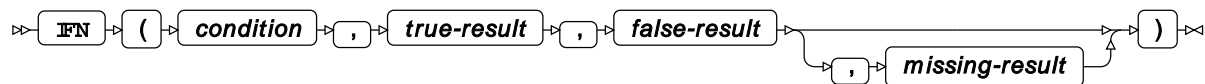
This produces the following output:

```
The string returned is: Missing
```

In this example, the `CHOOSEN` function contains an uninitialised variable (*var1*), which causes the function to return a missing value. In turn, this causes the `IFC` function to use the *missing-string* argument.

IFN

Returns a specified result based on a control condition.



Evaluates a control variable *condition*, and returns one of three specified values: *true-result* if *condition* is non-zero, *false-result* if it is zero, or *missing-result* if *condition* contains a missing value. If the argument *missing-result* is omitted, it is considered missing.

Note:

The variable *condition* is numeric, so any logical expressions used to determine its value have to return a number upon evaluation.

Return type: Numeric

condition

Type: Numeric

The control condition.

true-result

Type: Numeric

The value to return if the control condition evaluates to true.

false-result

Type: Numeric

The value to return if the control condition evaluates to false.

missing-result

Optional argument

Type: Numeric

The value to return if the control condition evaluates to missing.

Examples

In these examples, a specified result based on a control condition is returned. The results are written to the log.

```
DATA _NULL_;
  a1 = IFN("4",1,0,13);
  PUT a1=;
  a2 = IFN("four",1,0,13);
  PUT a2=;
  a3 = IFN("four",1,0);
  PUT a3=;
  a4 = IFN(0,1,0,13);
  PUT a4=;
RUN;
```

This produces the following output:

```
a1=1
a2=13
a3=.
a4=0
```

The argument value "4" has been converted into a number, but the argument value "four" is considered missing.

Cryptographic functions

Create cryptographic hashes.

You can create hashes (message digests) using MD5, SHA-1 and functions in the SHA-2 family.

MD5 ↗	675
Returns an MD5 message digest generated from specified text.	
PWENCODE ↗	676
Returns an encoded password.	
SHA1 ↗	677
Returns a SHA-1 message digest generated from specified text.	
SHA256 ↗	677
Returns a SHA-256 message digest generated from specified text.	

SHA384 ↗	678
Returns a SHA-384 message digest generated from specified text.	
SHA512 ↗	679
Returns a SHA-512 message digest generated from specified text.	
CALL AES256DEC ↗	680
Returns a text decoded from a ciphertext originally encrypted using Advanced Encryption Standard 256.	
CALL AES256ENC ↗	682
Returns a ciphertext created using Advanced Encryption Standard 256.	

MD5

Returns an MD5 message digest generated from specified text.

```
MD5 ( string )
```

The message digest is a 128-bit (16-byte) value. The digest is returned as binary, which can be displayed by applying the `$HEX.` format.

Return type: Character

string

Type: Character

The string for which an MD5 message digest is required.

Example

In this example, the function is used to create an MD5 message digest for a specified string. The result is written to the log.

```
DDATA _NULL_;

hk = MD5('mypasskey');
PUT 'The MD5 message digest is ' hk $HEX16.;

RUN;
```

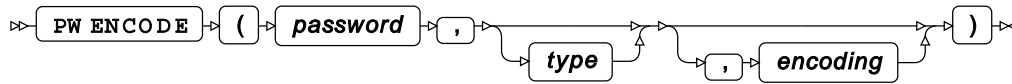
This produces the following output:

```
The MD5 message digest is 1414460444AE18E7
```

The value is returned as a binary, which is then converted to a character string using the `$HEX16.` format.

PW_ENCODE

Returns an encoded password.



Return type: Character

The encoded password.

password

Type: Character

The password.

type

Optional argument

Type: Numeric

1 or 3, for {sas001}, and {sas003} encryption code.

encoding

Optional argument

Type: Character

The name of an encoding to translate the password into, for example, 'UTF-8'.

Example

In this example, the function is configured to use the `sas001` encryption code and `UTF-8` encoding. The result is written to the log.

```
DATA _NULL_;  
    password = PW_ENCODE("BananasAndCustard", 1, 'UTF-8');  
    PUT password;  
RUN;
```

This produces the following output:

```
{sas001}QmFuYW5hc0FuZEN1c3RhcmQ=
```

SHA1

Returns a SHA-1 message digest generated from specified text.

➤ **SHA1** ➤ (➤ *text* ➤) ➤

The message digest is a 160-bit (20 byte) value. The digest is returned as binary, which can be displayed by applying the `$HEX.` format.

Return type: Character

text

Type: Character

The string for which an SHA-1 message digest is required.

Example

In this example, the function is used to create a SHA1 message digest for a supplied string. The result is written to the log.

```
DATA _NULL_;  
  
    hk = SHA1("mypasskey");  
    PUT "The SHA1 message digest is: " hk $HEX.;  
  
RUN;
```

This produces the following output:

```
The SHA1 message digest is: D18F7DCA9C9788AAC9B87878A11F369D5DADCBC6
```

The value is returned as a binary, which is then converted to a character string using the `$HEX.` format.

SHA256

Returns a SHA-256 message digest generated from specified text.

➤ **SHA256** ➤ (➤ *text* ➤) ➤

SHA-256 is one of the SHA-2 family of functions. A SHA-256 message digest is a 256-bit (32-byte) value. The digest is returned as binary, which can be displayed by applying the `$HEX.` format.

Return type: Character

text

Type: Character

The string for which a SHA-256 message digest is required.

Example

In this example, the function is used to create a SHA-256 message digest for a supplied string. The result is written to the log.

```
DATA _NULL_;  
  
    hk = SHA256("mypasskey");  
  
    PUT "The SHA256 message digest is: " hk $hex64.;  
  
RUN;
```

This produces the following output:

```
The SHA256 message digest is:  
523466C25030D8A6A9753D48E6A1072AB2C89453BBD2F44A75E3B116F70E74D7
```

The value is returned as a binary, which is then converted to a character string using the `$HEX.` format.

SHA384

Returns a SHA-384 message digest generated from specified text.



SHA-384 is one of the SHA-2 family of functions. A SHA-384 message digest is a 384-bit (48-byte) value. The digest is returned as binary, which can be displayed by applying the `$HEX.` format.

Return type: Character

text

Type: Character

The string for which a SHA-384 message digest is required.

Example

In this example, the function is used to create a SHA-384 message digest for a supplied string. The result is written to the log.

```
DATA _NULL_;  
  
    hk = SHA384("mypasskey");  
  
    PUT "The SHA384 message digest is: " hk $hex84.;  
  
RUN;
```

This produces the following output:

```
The SHA384 message digest is:  
5B07E5873C492924DE00A4076639E33CDE0BE381C0DBC9FC0EFB22FBDB2006CA6F90902DC50F2D1309FD  
DA0978FD6E9D
```

The value is returned as a binary, which is then converted to a character string using the `$HEX.` format.

SHA512

Returns a SHA-512 message digest generated from specified text.

⇒ **SHA512** ⇒ (⇒ *text* ⇒) ⇒ ⇐

SHA-512 is one of the SHA-2 family of functions. A SHA-512 message digest is a 512-bit (64-byte) value. The digest is returned as binary, which can be displayed by applying the `$HEX.` format.

Return type: Character

text

Type: Character

The string for which an SHA-512 message digest is required.

Example

In this example, the function is used to create a SHA-512 message digest for a supplied string. The result is written to the log.

```
DATA _NULL_;  
  
hk = SHA512("mypasskey");  
  
PUT "The SHA512 message digest is: " hk $hex128.;  
  
RUN;
```

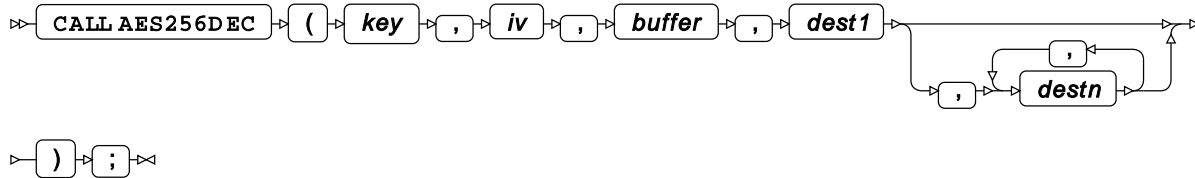
This produces the following output:

```
The SHA512 message digest is:  
8F4CBD1050028B6748176EDA9111483464744FD76E099B54D60A030E94A9D43A92CBFAE06E1E8CB2ED70  
B374E59CEDB64A56D22679745CC19AB333B8DADAB8B5
```

The value is returned as a binary, which is then converted to a character string using the `$HEX.` format.

CALL AES256DEC

Returns a text decoded from a ciphertext originally encrypted using Advanced Encryption Standard 256.



Advanced Encryption Standard (AES) 256 is an industry-standard encryption algorithm. For information on AES 256, see [Announcing the Advanced Encryption Standard \(AES\)](#). This function uses AES counter (CTR) mode.

Information can be encrypted as AES 256 using `CALL AES256ENC`. [\(page 680\)](#)

To decrypt a ciphertext encrypted using AES 256 you need the key and initialisation vector used to encrypt the source information.

key

Type: Character

The key that was used to encrypt the source data. This must be a 32-byte string.

The key can be a string created using `PWENCODE` [\(page 676\)](#).

iv

Type: Character

The initialisation vector that was used to encrypt the source data. This must be a 16-byte string.

buffer

Type: Var

A variable or array item that contains the ciphertext (that is, the encrypted data).

dest1

Type: Var

A variable or array item that will contain the decrypted information.

destn

Optional argument

Type: Var

A variable or array item that will contain the decrypted information.

You can specify that the ciphertext is decrypted into more than one destination item. The ciphertext might have been constructed from many source items, and you might want to decrypt to the same number of items. You can therefore specify more than one destination (*dest1* ... *destn*) for the decryption.

Basic example

In this example, a string is decrypted. The result is written to the log.

```
data _null_;

  format pt $18.;

  ct = "0014F19A0DBB99066680870EBAB99950762FC58E"x;

  call aes256dec("HVZ5T68AE1WFM89GXFKC236IMV7L208Z",
    "ivvvvVVVVvvVVvVi", ct, pt);

  put "The decrypted text is: " pt;

run;
```

This produces the following output:

```
The decrypted text is: I have been hidden
```

Example – more than one source

In this example, three strings are decrypted. The result is written to the log.

```
data _null_;

  format pt1 $18.;
  format pt2 $14.;
  format pt3 $18.;

  ct = "0034F19A0DBB99066680870EBAB99950762FC58E339D2E3EEBD8C4C8D5BE1
034C905AC8DE63FF65C73DD4281B128FBE2AF8634D3"x;

  call aes256dec("HVZ5T68AE1WFM89GXFKC236IMV7L208Z",
    "ivvvvVVVVvvVVvVi", ct, pt1, pt2, pt3);

  put "The decrypted text is: ";
  put pt1;
  put pt2;
  put pt3;

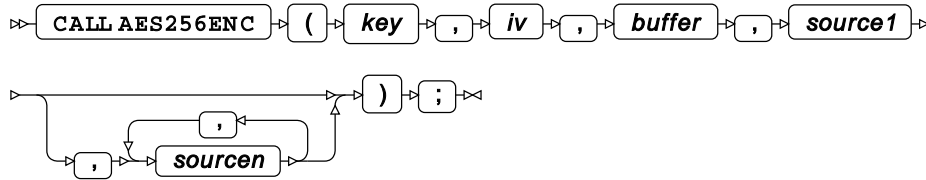
run;
```

This produces the following output:

```
The decrypted text is:
I have been hidden
AES 256 hid me
Now I am decrypted
```

CALL AES256ENC

Returns a ciphertext created using Advanced Encryption Standard 256.



Advanced Encryption Standard (AES) 256 is an industry-standard encryption algorithm. For information on AES 256, see [Announcing the Advanced Encryption Standard \(AES\)](#). This function uses AES counter (CTR) mode.

Information encrypted using this routine can be decrypted using `CALL AES256DEC` ([page 680](#)).

key

Type: Character

The key to be used to encrypt the source data. This must be a 32-byte string.

The key can be a string created using `PWENCODE` ([page 676](#)).

iv

Type: Character

The initialisation vector to apply. This must be a 16-byte string.

buffer

Type: Var

A variable or array item that will contain the ciphertext (that is, the encrypted data).

source1

Type: Var

A variable or array item that contains data to be encrypted.

sourcen

Optional argument

Type: Var

A variable or array item that contains data to be encrypted.

You can specify more than one source of information; these will be concatenated to create one ciphertext.

Basic example

In this example, a string is encrypted. The result is written to the log.

```
data _null_;

    format ct $HEX64.;
    pt = "text to encrypt";

    call aes256enc("HVZ5T68AE1WFM89GXFKC236IMV7L208Z", "ivvvvVVVVvvVVvVi", ct, pt);

    put "The encrypted string is: " ct;

run;
```

This produces the following output:

```
The encrypted string is: 0011CCDF1DAECF1729C28705B7EB884966
```

Example – more than one source

In this example, three strings are encrypted. The result is written to the log.

```
data _null_;

    format ct $HEX134.;
    pt1 = "text to encrypt";
    pt2 = "another text to encrypt";
    pt3 = "and another text to encrypt";

    call aes256enc("HVZ5T68AE1WFM89GXFKC236IMV7L208Z",
                  "ivvvvVVVVvvVVvVi", ct, pt1, pt2, pt3);

    put "The encrypted string is: " ct;

run;
```

This produces the following output:

```
The encrypted string is:
0043CCDF1DAECF1729C28705B7EB8849662ACE8F06B0186CF9999790C9F7007B84058C81E366CF0873DE0
6C5B525E6EFB797239721382308070FB356A3F2F0809DF4F4
```

Date and time functions and CALL routine

Select, convert and calculate dates and times; for example, you can convert dates to Julian dates, get the current date, count completed time intervals, or return the weekday number.

Dates and datetimes are stored in WPS as numeric values, using midnight on 01-January-1960 as a reference date (epoch). From this reference point, each increment in a date value represents one day from epoch, and each increment in a datetime value represents one second from midnight of the same day. Times are stored as a numeric value, using midnight as the reference point, with each increment representing one second.

WPS can represent dates between 01-January-1582 (date value of -138061) and 31-December-9999 (date value of 2936547); the equivalent datetime limits are between 01-January-1582:00:00:00 (datetime value of -11928470400) and 31-December-9999:23:59:59 (datetime value of 253717747199).

Time values should be less than 86400 (the number of seconds in a day); any numeric value greater than this interpreted as a time value returns the total number of hours, minutes and seconds.

You need to be aware of the type of value you are using as the format applied to the stored numeric can change the output. For example, the following uses a time value (`tValue`) that represents 12:30:45 to which time, date and datetime formats can be applied:

```
DATA _NULL_;
  tValue = 45045;
  PUT "Number as Time:      " tValue time.;
  PUT "Number as Date:      " tValue date11.;
  PUT "Number as DateTime:  " tValue datetime.;
RUN;
```

When the time and datetime formats are applied, similar output for the time in the day is generated; the date value is unlikely to be accurate:

```
Number as Time:      12:30:45
Number as Date:      30-APR-2083
Number as DateTime:  01JAN60:12:30:45
```

Date and time literals

As an alternative to using the numeric value for a date or datetime, you can append date or time identifiers to a string, and the string will be treated as a date, time or datetime value. The modifiers appended to the strings (after any quotes) are:

T

When appended to a string, the content of the string is interpreted as a time value.

```
DATA _NULL_;
  tm = "12:30:45"T;
  PUT "Time value:          " tm;
  PUT "Time formatted:      " tm time.;
RUN;
```

Output as the numeric value and a formatted equivalent:

```
Time value:      45045
Time formatted:  12:30:45
```

D

When appended to a string, the content of the string is interpreted as a date value.

```
DATA _NULL_;
  da = "29-SEP-2014"D;
  PUT "Date value:      " da;
  PUT "Date formatted:  " da date11.;
RUN;
```

Output as the numeric value and a formatted equivalent:

```
Date value:      19995
Date formatted:  29-SEP-2014
```

DT

When appended to a string, the content of the string is interpreted as a datetime value.

```
DATA _NULL_;
  dm = "29-SEP-2014:12:30:45"DT;
  PUT "Datetime value:    " dm;
  PUT "Datetime formatted: " dm datetime.;
RUN;
```

Output as the numeric value and a formatted equivalent:

```
Datetime value:    1727613045
Datetime formatted: 29SEP14:12:30:45
```

Date and time functions and CALL routines are used to generate *date*, *datetime* and *time* values; select part of a value, convert values, or calculate differences between *date* or *datetime* values.

Get date and time values ↗	686
DATA step functions that generate <i>date</i> and <i>datetime</i> values from epoch; generate the time value within the current day; or convert date and time constants to a date, datetime or time value.	
Select date and time values ↗	693
DATA step functions that select part of a <i>date</i> value, <i>datetime</i> value, or <i>time</i> value.	
Convert date and time values ↗	701
DATA step functions and CALL routine that convert between <i>date</i> values and Julian dates, or create <i>date</i> or <i>time</i> values conforming to the ISO 8601:2004 standard.	
Calculate date and time values ↗	709
DATA step functions that calculate differences between <i>date</i> and <i>datetime</i> values; or increment date and datetime values to create new date and datetime values.	

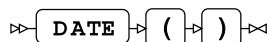
Get date and time values

DATA step functions that generate *date* and *datetime* values from epoch; generate the time value within the current day; or convert date and time constants to a date, datetime or time value.

DATE ↗	686
Returns the count of days since epoch. DATE is an alias of TODAY.	
TODAY ↗	687
Returns the date value for the current day. TODAY is an alias of DATE	
DATETIME ↗	687
Returns the number of seconds since epoch for the current time.	
TIME ↗	688
Returns the current time as the number of seconds from midnight.	
MDY ↗	689
Returns a date value for a specified month–day–year combination.	
DHMS ↗	690
Returns a datetime value for a specified date and time (hour, minute, second) combination.	
HMS ↗	691
Returns a time value for a specified hour–minute–second combination.	
YYQ ↗	692
Returns the date value for the first day of a quarter from a specified year and quarter.	

DATE

Returns the count of days since epoch. DATE is an alias of TODAY.



This function takes no arguments and returns the total number of days from epoch

Return type: Numeric

Example

The following example returns the date value for the current day and prints the output as a numeric value. The date value is also reformatted to display as YYYY-MM-DD using the `YYMMDD10.` format.

```

DATA _NULL_;
  d = DATE();
  PUT "Today:           " d;
  PUT "Default format:  " d DATE.;
  PUT "YYYY-MM-DD format: " d YYMMDD10.;
RUN;

```

Which produces the following output in the log:

```
Today:          20514
Default format: 01MAR16
YYYY-MM-DD format: 2016-03-01
```

TODAY

Returns the date value for the current day. `TODAY` is an alias of `DATE`

➤ `TODAY` ➤ () ➤

This function takes no arguments and returns the total number of days from epoch.

Return type: Numeric

Example

The following example returns the date value for the current day and prints the output as a numeric value. The date value is also reformatted to display as YYYY-MM-DD using the `YYMMDD10.` format.

```
DATA _NULL_;
  d = TODAY();
  PUT "Today: " d;
  PUT "Default format: " d date.;
  PUT "YYYY-MM-DD format: " d YYMMDD10.;
RUN;
```

Which produces the following output in the log:

```
Today: 20571
Default format: 27APR16
YYYY-MM-DD format: 2016-04-27
```

DATETIME

Returns the number of seconds since epoch for the current time.

➤ `DATETIME` ➤ () ➤

This function takes no arguments and returns the count of seconds for the local time. The value returned can contain up to three decimal places, representing the millisecond count for the current second, and can be used as an argument to other date and time functions.

Return type: Numeric

Example

The following example prints the current datetime in the numeric format and the same date time in a datetime format that displays the milliseconds in the datetime value as the fractional part of the decimal number of seconds

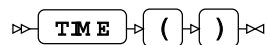
```
DATA _NULL_;  
  dt = datetime();  
  PUT "Time now:          " dt;  
  PUT "Default format: " dt datetime24.5;  
RUN;
```

Which produces the following output in the log:

```
Time now:          1781097234.6  
Default format: 09JUN2016:13:13:54.61000
```

TIME

Returns the current time as the number of seconds from midnight.



This function takes no arguments and returns the count of seconds from midnight for the current day.

Return type: Numeric

Example

The following example returns the current time and prints the output as a numeric value. The time value is also reformatted to display using the default `TIME.` format.

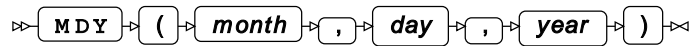
```
DATA _NULL_;  
  tM = TIME();  
  PUT "Time in seconds " tM;  
  PUT "Current time: " tM TIME.;  
RUN;
```

Which produces the following output in the log:

```
Time in seconds 35894.681  
Current time: 9:58:15
```

MDY

Returns a date value for a specified month–day–year combination.



Return type: Numeric

The function will only return a date where a combination of month–day–year is valid.

month

Type: Numeric

The numeric representation of the month within the year.

day

Type: Numeric

The numeric representation of the day within the month.

year

Type: Numeric

A two or four digit representation of the year. The two-digit year range begins at the year set in the `YEARCUTOFF` system option; by default the range of two-digit dates is 01-January-1926 through to 31-December-2025.

Example

The following example returns the date value for the month–day–year value 8–17–2015 and prints the output as a numeric value. The date value is also reformatted to display using the `DATE11.` format.

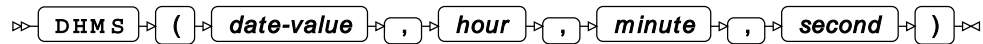
```
DATA _NULL_;
  hour1 = MDY(8, 17, 2015);
  PUT "DATE value:      " hour1;
  PUT "Formatted date: " hour1 DATE11.;
RUN;
```

Which produces the following output:

```
DATE value:      20317
Formatted date: 17-AUG-2015
```

DHMS

Returns a datetime value for a specified date and time (hour, minute, second) combination.



All arguments must be supplied to this function. Each of the numeric hour, minute, and second fields do not have an upper limit allowing you to calculate a future datetime value from midnight of the date argument as the start point.

Return type: Numeric

date-value

Type: Numeric

The base date value from which the datetime value is calculated. This can be a known date value, a date literal, or calculated using other functions that return a date value.

hour

Type: Numeric

The number of hours to be added to the *date-value*. Typically the number would be in the range 0–23.

minute

Type: Numeric

The number of minutes to be added to the *hour* value. Typically the number would be in the range 0–59.

second

Type: Numeric

The number of seconds to be added to the *minute* value. Typically the number would be in the range 0–59.

Example

This example calculates two datetime values; one representing midnight on the start date, and one representing a datetime value 100 hours, 100 minutes and 100 seconds after the start date value.

```
DATA _NULL_;
  datetime1 = DHMS ("17-AUG-2015"d, 0, 0, 0);
  datetime2 = DHMS ("17-AUG-2015"d, 100, 100, 100);
  PUT "Formatted output: " datetime1 DATETIME.;
  PUT "Future date output: " datetime2 DATETIME.;
RUN;
```


Which produces the following output in the log.

```
Formatted output: 17AUG15:00:00:00
Future date output: 21AUG15:05:41:40
```

Example

The `DHMS` function can also be used to convert date and time values into a single datetime value, or generate a datetime value for the current date and time:

```
DATA _NULL_;
  td = TODAY();
  tm = TIME();
  dtNow = DHMS(td, HOUR(tm), MINUTE(tm), SECOND(tm));
  PUT dtNow DATETIME.;
RUN;
```

Which produces the following output in the log.

```
17AUG15:13:26:40
```

HMS

Returns a time value for a specified hour–minute–second combination.

➤ **HMS** ➤ (➤ *hour* ➤ , ➤ *minute* ➤ , ➤ *second* ➤) ➤ ⏏

All arguments must be supplied to this function. A time value is calculated from the supplied number of hours, minutes and seconds, and will return the time with reference to midnight on the current day. The arguments do not have an upper limit allowing you to calculate a future time value as well as time within the current day.

Return type: Numeric

hour

Type: Numeric

The number of hours from midnight of the current day. Typically the number would be in the range 0–23.

minute

Type: Numeric

The number of minutes to be added to the *hour* value. Typically the number would be in the range 0–59.

second

Type: Numeric

The number of seconds to be added to the *minute* value. Typically the number would be in the range 0–59.

Example

The following example returns the time value from a number of hours, minutes and seconds from midnight of the current day

```
DATA _NULL_;  
  time1 = HMS (12, 15, 45);  
  time2 = HMS (100, 100, 100);  
  PUT "Time1 output:           " time1 time.;  
  PUT "Time2 output as time: " time2 time.;  
RUN;
```

Which produces the following output in the log

```
Time1 output:           12:15:45  
Time2 output as time:   101:41
```

YYQ

Returns the date value for the first day of a quarter from a specified year and quarter.

⇒ **YYQ** ⇒ (⇒ **year** ⇒ , ⇒ **quarter** ⇒) ⇒

Return type: Numeric

year

Type: Numeric

A two or four digit representation of the year. The two-digit year range begins at the year set in the `YEARCUTOFF` system option; by default the range of two-digit dates is 01-January-1926 through to 31-December-2025.

quarter

Type: Numeric

The quarter within the year, in the range 1–4.

Example

The following example uses a DO loop to find the first day in each quarter of a year.

```
DATA _NULL_;
  DO i=1 to 4;
    Qt = YYQ(2015, i);
    PUT "First day of quarter " i ": " Qt DATE11.;
  END;
RUN;
```

Which produces the following output in the log.

```
First day of quarter 1 : 01-JAN-2015
First day of quarter 2 : 01-APR-2015
First day of quarter 3 : 01-JUL-2015
First day of quarter 4 : 01-OCT-2015
```

Select date and time values

DATA step functions that select part of a *date* value, *datetime* value, or *time* value.

DATEPART ↗	694
Returns the date value from a datetime value.	
TIMEPART ↗	694
Returns the time section of a datetime value,	
YEAR ↗	695
Returns the year numeric value from a date.	
QTR ↗	695
Returns the quarter in the year a date value falls within.	
MONTH ↗	696
Returns the month numeric value from a date.	
WEEK ↗	697
Returns the week count in the year within which the specified date occurs.	
WEEKDAY ↗	698
Returns the weekday number from a date value.	
DAY ↗	699
Returns the day number within the month from a date value.	
HOUR ↗	699
Returns the hour value from a datetime or time value.	
MINUTE ↗	700
Returns the number of minutes from a datetime or time value.	
SECOND ↗	701
Returns the number of seconds from a datetime or time value.	

DATEPART

Returns the date value from a datetime value.



This function can be used to convert a datetime value into a date value.

Return type: Numeric

datetime-value

Type: Numeric

The datetime value from which the date value will be returned. This can be a known datetime, a datetime literal, or calculated using other functions.

Example

The following example converts a datetime value into a date format (DD-MMM-YYYY), and displays the output using the DATE11. format.

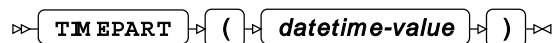
```
DATA _NULL_;  
  dpart=DATEPART("17-AUG-2015:12:30:45"dt);  
  PUT "Date from a datetime: " dpart DATE11.;  
RUN;
```

Which produces the following output in the log:

```
Date from a datetime: 17-AUG-2015
```

TIMEPART

Returns the time section of a datetime value,



Return type: Numeric

datetime-value

Type: Numeric

The datetime value from which the time is returned. This can either be a known datetime value, or calculated using other functions that return a datetime value.

Example

```
DATA _NULL_;  
  dtM = TIMEPART ("17-AUG-2015:12:30:45"DT);  
  PUT "Time: " dtM TIME.;  
RUN;
```

Which produces the following output in the log:

```
Time: 12:30:45
```

YEAR

Returns the year numeric value from a date.

```
>> [YEAR] -> ( -> [date-value] -> ) <<
```

Return type: Numeric

date-value

Type: Numeric

The date value from which the year part is returned. This can be a known date value, a date literal, or calculated using other functions that return a date value.

Example

The following example returns the year value from a date literal.

```
DATA _NULL_;  
  Hy = "17-AUG-2015"D;  
  Yr = YEAR(Hy);  
  PUT "Year: " Yr;  
RUN;
```

Which produces the following output in the log:

```
Year: 2015
```

QTR

Returns the quarter in the year a date value falls within.

```
>> [QTR] -> ( -> [date-value] -> ) <<
```

Return type: Numeric

date-value

Type: Numeric

The date value for which the quarter is calculated. This can be a known date value, a date literal, or the result of using other functions that return a date value.

Example

The following example returns the quarter value from a date literal.

```
DATA _NULL_;  
  Hy = "17-AUG-2015"D;  
  Qt = QTR(Hy);  
  PUT "Date is in quarter: " Qt;  
RUN;
```

Which produces the following output in the log

```
Date is in quarter: 3
```

MONTH

Returns the month numeric value from a date.

➤ MONTH ➤ (➤ ***date-value*** ➤) ➤

The returned numeric represents the month in the year, where 1=January, 2=February and so on.

Return type: Numeric

date-value

Type: Numeric

The date value from which the month part is returned. This can either be a known date value, a date literal, or calculated using other functions that return a date value.

Example

The following example returns the month as a numeric value; a string representation of the month can be returned by formatting the date value using `MONNAME`.

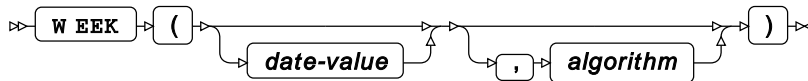
```
DATA _NULL_;  
  Hy = "17-AUG-2015"D;  
  Mn = MONTH(Hy);  
  PUT "Month from DATE value: " Mn;  
  PUT "Formatted month:      " Hy MONNAME.;  
RUN;
```

Which produces the following output in the log:

```
Month from DATE value: 8
Formatted month:      August
```

WEEK

Returns the week count in the year within which the specified date occurs.



Where no *date-value* is provided to the function, the week value for `TODAY` is returned; if no *algorithm* is specified, the "U" method is used.

Return type: Numeric

date-value

Optional argument

Type: Numeric

The date value for which the week count is determined. This can be a known date value, a date literal, or calculated using other functions that return a date value.

algorithm

Optional argument

The method to be used when calculating the week number.

"U"

Specifies the first day of the week is Sunday. At the start of the year, any dates that fall before the first Sunday of the year have a week count of "0".

"V"

Calculates the week based on the *ISO 8601 Data elements and interchange formats – Information interchange – Representation of dates and times* standard definition for the number of weeks in a year and the first week of the year.

The standard defines the first day of the week as a Monday, and the first week in a year is that week that contains the first Thursday occurring in the year. At the start of the year, any dates falling before week one will be either week 52 or week 53 (if the year is a long year) of the previous year.

"W"

Specifies the first day of the week is Monday. At the start of the year, any dates that fall before the first Monday of the year have a week count of "0"

Example

The following example returns the week count for the current day, and the week count of a known date. Both functions use the "U" calculation method.

```
DATA _NULL_;
  Td = TODAY();
  Wk  = WEEK();
  Hy  = "17-AUG-2015"D;
  HyWK = WEEK(Hy, "U");
  PUT "Today's date:           " Td DATE11.;
  PUT "Week count of today:    " Wk;
  PUT "Week count of Hy Date: " HyWk;
RUN;
```

Which produces the following output in the log.

```
Today's date:           17-AUG-2016
Week count of today:    33
Week count of Hy Date: 33
```

WEEKDAY

Returns the weekday number from a `date value`.

```
WEEKDAY ( date-value )
```

The returned numeric represents the day in the week, where 1=Sunday, 2=Monday and so on.

Return type: Numeric

date-value

Type: Numeric

The date value from which the weekday part is returned. This can either be a known date value, a date literal, or calculated using other functions that return a date value.

Example

The following example returns the weekday as a numeric value; a string representation of the weekday can be returned by formatting the date value using `DOWNNAME`.

```
DATA _NULL_;
  Hy = "17-AUG-2015"D;
  WK = WEEKDAY(Hy);
  PUT "Weekday number: " WK;
  PUT "Weekday name: " Hy DOWNNAME.;
RUN;
```


Which produces the following output in the log:

```
Weekday number: 2  
Weekday name:   Monday
```

DAY

Returns the day number within the month from a date value.



Return type: Numeric

date-value

Type: Numeric

The date from which the day in the month is returned. This can be a known date value, a date literal, or calculated using other functions that return a date value.

Example

The following example returns the day count for specified date value.

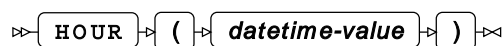
```
DATA _NULL_;  
  DAY1 = DAY("17-AUG-2015"D);  
  PUT "Day count from DAY1 date: " DAY1;  
RUN;
```

Which produces the following output in the log:

```
Day count from DAY1 date: 17
```

HOUR

Returns the hour value from a datetime or time value.



Return type: Numeric

datetime-value

Type: Numeric

The datetime or time value from which the hour is returned. This can either be a known datetime or time value, or calculated using other functions that return a datetime or time value.

Example

The following example returns the hour count for specified datetime value.

```
DATA _NULL_;  
  TIME1 = HOUR("17-AUG-2015:12:30:45"DT);  
  PUT "Hour part of the time: " TIME1;  
RUN;
```

Which produces the following output

```
Hour part of the time: 12
```

MINUTE

Returns the number of minutes from a datetime or time value.

➤ **MINUTE** ➤ (➤ *datetime-value* ➤) ➤

Return type: Numeric

datetime-value

Type: Numeric

The datetime or time value from which the minute count is extracted. This can either be a known datetime or time value, or calculated using other functions that return a datetime or time value.

Example

The following example returns the minute count for specified datetime value.

```
DATA _NULL_;  
  Mn=MINUTE("17-AUG-2015:12:30:45"DT);  
  PUT "Number of minutes: " Mn;  
RUN;
```

Which produces the following output in the log:

```
Number of minutes: 30
```

SECOND

Returns the number of seconds from a datetime or time value.



Return type: Numeric

datetime-value

Type: Numeric

The datetime or time value from which the count of seconds is returned. This can either be a known datetime or time value, or calculated using other functions that return a datetime or time value

Example

The following example returns the second count for specified datetime value.

```

DATA _NULL_;
  SEC1 = SECOND("17-AUG-2015:12:30:45"DT);
  PUT SEC1;
RUN;
  
```

Which produces the following output in the log:

```

Number of seconds: 45
  
```

Convert date and time values

DATA step functions and CALL routine that convert between *date* values and Julian dates, or create *date* or *time* values conforming to the ISO 8601:2004 standard.

DATEJUL ↗	702
Converts a five-digit or seven-digit Julian Date into a date value.	
JULDATE ↗	703
Converts a date value into a Julian Date.	
JULDATE7 ↗	704
Converts a date value into a seven-digit (YYYYDDD) Julian Date.	
CALL IS8601_CONVERT ↗	704
Converts date and datetime formats to and from ISO 8601 time intervals and durations.	

DATEJUL

Converts a five-digit or seven-digit Julian Date into a date value.

➤ **DATEJUL** ➤ (➤ *julian-date* ➤) ➤

In five-digit Julian Date, the first two digits represent the year starting at the year set in the `YEARCUTOFF` system option; by default the range of two-digit dates is 01-January-1926 through to 31-December-2025. In a seven-digit Julian Date, the first four digits represent the year. In both five-digit and seven-digit Julian Dates, the final three digits represent the count of days in the year.

Return type: Numeric

When using Julian Dates in SAS language programs, we recommend using a seven-digit format to ensure there is no confusion in date values.

julian-date

Type: Numeric

The five-digit (YYDDD) or seven-digit (YYYYDDD) Julian Date to convert to a SAS language date numeric.

Example

The following example converts a five-digit and seven-digit Julian date into a date value. The `YEARCUTOFF` is first printed to see whether the returned date value is the same.

```
PROC OPTIONS OPTION=YEARCUTOFF;
RUN;

DATA _NULL_ ;
  d5 = 30061;
  d7 = 2030061;
  jdate1 = DATEJUL(d5);
  jdate2 = DATEJUL(d7);
  PUT "From five-digit date: " jdate1 DATE11.;
  PUT "From seven-digit date: " jdate2 DATE11.;
RUN;
```

Which produces the following output in the log:

```
YEARCUTOFF=1926      Cutoff year used when interpreting or generating 2 digit
                      years in functions and formats
From five-digit date: 02-MAR-1930
From seven-digit date: 02-MAR-2030
```

JULDATE

Converts a date value into a Julian Date.

➤ **JULDATE** ➤ (➤ *date-value* ➤) ➤

In five-digit Julian Date, the first two digits represent the year starting at the year set in the `YEARCUTOFF` system option; by default the range of two-digit dates is 01-January-1926 through to 31-December-2025. In a seven-digit Julian Date, the first four digits represent the year. In both five-digit and seven-digit Julian Dates, the final three digits represent the count of days in the year.

For dates falling in the one hundred years after the 01-January in the `YEARCUTOFF`, this function returns the Julian Date in five-digit format; for all other dates this function returns the Julian Date in seven-digit format.

Return type: Numeric

date-value

Type: Numeric

The date value to be converted to a Julian Date. This can be a known date value, a date literal, or calculated using other functions that return a date value.

Example

The following example returns either a five-digit or seven-digit Julian date value depending upon whether the date is within 100 years of the `DATECUTOFF` value.

```
PROC OPTIONS OPTION=YEARCUTOFF;
RUN;

DATA _NULL_;
  JDATE1 = JULDATE("31-DEC-1925"D);
  JDATE2 = JULDATE("01-JAN-1926"D);
  JDATE3 = JULDATE("31-DEC-2025"D);
  JDATE4 = JULDATE("01-JAN-2026"D);
  PUT "JDATE1: " JDATE1;
  PUT "JDATE2: " JDATE2;
  PUT "JDATE3: " JDATE3;
  PUT "JDATE4: " JDATE4;
RUN;
```

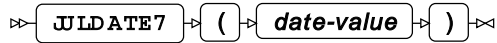
Which produces the following output in the log:

```
YEARCUTOFF=1926      Cutoff year used when interpreting or generating 2 digit
                     years in functions and formats

JDATE1: 1925365
JDATE2: 26001
JDATE3: 25365
JDATE4: 2026001
```

JULDATE7

Converts a date value into a seven-digit (YYYYDDD) Julian Date.



Return type: Numeric

date-value

Type: Numeric

The date value to be converted to a Julian Date. This can be a known date value, a date literal, or calculated using other functions that return a date value.

Example

```

DATA _NULL_;
  JDATE1 = JULDATE7 ("31-DEC-1919"d);
  JDATE2 = JULDATE7 ("01-JAN-1920"d);
  JDATE3 = JULDATE7 ("31-DEC-2019"d);
  JDATE4 = JULDATE7 ("01-JAN-2020"d);
  PUT "JDATE1: " JDATE1;
  PUT "JDATE2: " JDATE2;
  PUT "JDATE3: " JDATE3;
  PUT "JDATE4: " JDATE4;
RUN;

```

Which produces the following output in the log:

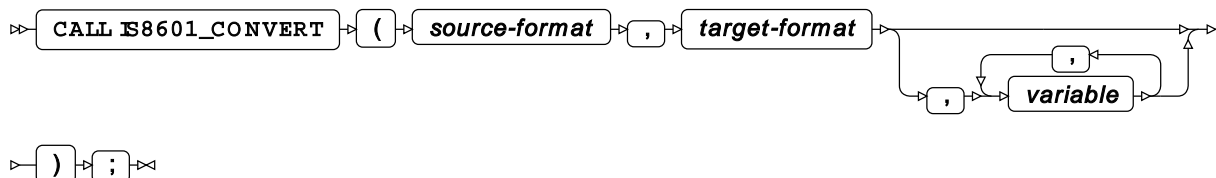
```

JDATE1: 1919365
JDATE2: 1920001
JDATE3: 2019365
JDATE4: 2020001

```

CALL IS8601_CONVERT

Converts date and datetime formats to and from ISO 8601 time intervals and durations.



The number of input and returned values are determined by the settings used in the `source-format` and `target-format` arguments. These arguments are followed by any input values or return value placeholders in the number and order defined in the format arguments.

Time intervals and durations are described in the standard, *Data elements and interchange formats – Information interchange – Representation of dates and times* standard. An ISO 8601 time interval describes the intervening time between two known time points; this can either be using the start and end points, or one of the known time points and a duration. An ISO 8601 duration describes a period of time without the context of a start or end date.

When using ISO 8601 dates and times, WPS uses an internal representation for both time intervals and durations. To present this information in output in a human-readable form, apply one of the formats described in ISO 8601 date formats [↗](#) (page 476)

source-format

Specifies the format, or formats, of the source date and time information, one of the following values:

"INTVL"

The input argument at position three is an ISO 8601 time interval.

"DT/DT"

Two input arguments are supplied, both are datetime values. The arguments at positions three and four are assumed to be datetime values and can be known datetime numeric values, datetime literals, or calculated using other functions that return a datetime value.

"DU/DT"

Two input arguments are provided; the first is an ISO 8601 duration, the second a datetime value. The input argument at position three must be a valid ISO 8601 time interval. The input argument at position four is assumed to be a datetime value, and can be a known datetime numeric value, a datetime literal, or calculated using other functions that returns a datetime value.

"DU/D"

Two input arguments are provided; the first is an ISO 8601 duration, the second a date value. The input argument at position three must be a valid ISO 8601 time interval. The input argument at position four is assumed to be a date value, and can be a known date numeric value, a date literal, or calculated using other functions that returns a date value.

"DT/DU"

Two input arguments are provided; the first is a datetime value, the second an ISO 8601 duration. The input argument at position three is assumed to be a datetime value, and can be a known datetime numeric value, a datetime literal, or calculated using other functions that returns a datetime value. The input argument at position four must be a valid ISO 8601 time interval.

"D/DU"

Two input arguments are provided; the first is a date value, the second an ISO 8601 duration. The input argument at position three is assumed to be a date value, and can be a known date numeric value, a string literal, or calculated using other functions that returns a date value. The input argument at position four must be a valid ISO 8601 time interval.

"D/D"

Two input arguments are supplied, both are date values. The arguments at positions three and four are assumed to be date values and can be known date numeric values, date literals, or calculated using other functions that return a date value.

"DT"

The input argument supplied is a datetime value. The argument at position three is assumed to be a datetime value and can be a known datetime numeric value, a datetime literal, or calculated using other functions that return a datetime value.

"DU"

The input argument at position three is an ISO 8601 duration.

"D"

The input argument supplied is a date value. The argument at position three is assumed to be a date value and can be a known date numeric value, a date literal, or calculated using other functions that return a date value.

target-format

Specifies the format, or formats, the resultant date and time information is converted into. `target-format` contains two options for calculating date and time values: "START" and "END". These options can be used with ISO 8601 intervals, and duration values when used with a date or datetime value.

"INTVL"

The output argument is an internal representation of the ISO 8601 interval format. To present this information, apply one of the formats described in ISO 8601 date formats [\(page 476\)](#)

"DT/DT"

Two datetime output arguments are returned.

"DU/DT"

Two output arguments are returned in the specified order; the first is an ISO 8601 duration, the second a datetime value.

"DU/D"

Two output arguments are returned in the specified order; the first is an ISO 8601 duration, the second a date value.

"DT/DU"

Two output arguments are returned in the specified order; the first is a datetime value, the second an ISO 8601 duration.

"D/DU"

Two output arguments are returned in the specified order; the first is a date value, the second an ISO 8601 duration.

"D/D"

Two date output arguments are returned.

"DT"

The output argument is a datetime value.

"DU"

The output argument is an internal representation of the ISO 8601 duration format. To present this information, apply one of the formats described in ISO 8601 date formats [\(page 476\)](#)

"D"

The output argument is a date value.

"START"

The output is a datetime value marking the start point of either an interval value or duration. The value is calculated from either an ISO 8601 Interval value, or a combination of duration and end date.

"END"

The output is a datetime value marking the end point of either an interval value or duration. The value is calculated from either an ISO 8601 Interval value, or a combination of duration and start date.

variable

Optional argument

Type: Character or numeric value

Arguments containing the input and output values. The number and order of the arguments is defined in the `source-format` and `target-format`.

Example – Create ISO 8601 intervals

This example shows how to create ISO 8601 standard intervals from date or datetime values. The format of the output intervals follows the structure defined in the standard; in the case of date values the time element is set to midnight (00:00:00):

```
DATA _NULL_;
  startDT = "17-AUG-2015:12:30:45"DT;
  startD   = "17-AUG-2015"D;
  endDT    = "18-SEP-2016:15:45:00"DT;
  endD     = "18-SEP-2016"D;
  FORMAT IntOutDT IntOutD $N8601EA.;
  CALL IS8601_CONVERT("DT/DT", "INTVL", startDT, endDT, IntOutDT);
  CALL IS8601_CONVERT("D/D", "INTVL", startD, endD, IntOutD);
  PUT "Interval from date:      " IntOutD;
  PUT "Interval from datetime: " IntOutDT;
RUN;
```

Which produces the following output in the log:

```
Interval from date:      2015-08-17T00:00:00/2016-09-18T00:00:00
Interval from datetime: 2015-08-17T12:30:45/2016-09-18T15:45:00
```

Example – Convert datetime to date values

In addition to converting to ISO8601 formats, the routine can be used to convert between datetime and date values. The following mimics using the DATEPART function to return dates from datetime values:

```
DATA _NULL_;
  startDT = "17-AUG-2015:12:30:45"DT;
  endDT    = "18-SEP-2016:15:45:00"DT;
  FORMAT outD1 outD2 DATE11.;
  CALL IS8601_CONVERT("DT/DT", "D/D", startDT, endDT, outD1, outD2);
  PUT "Date from startDT: " outD1;
  PUT "Date from endDT:  " outD2;
RUN;
```

Which produces the following output in the log:

```
Date from startDT: 17-AUG-2015
Date from endDT:  18-SEP-2016
```

Example – Finding start or end dates

This example shows how to use the routine to calculate dates, in this case how to find the end datetime where a start date and duration is known, and a start datetime where a duration and end date is known.

```
DATA _NULL_;
  startDT    = "17-AUG-2015:12:30:45"DT;
  endDT      = "18-SEP-2016:15:45:00"DT;
  duStartEnd = 33794055;
  CALL IS8601_CONVERT("DU/DT", "START", duStartEnd, endDT, StartDate);
  CALL IS8601_CONVERT("DT/DU", "END", startDT, duStartEnd, EndDate);
  PUT "Start Date: " StartDate DATETIME.;
  PUT "End Date:   " EndDate DATETIME.;
RUN;
```

Which produces the following output in the log:

```
Start Date: 17AUG15:12:30:45
End Date:   18SEP16:15:45:00
```

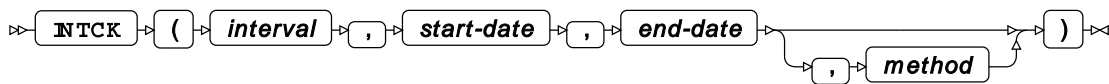
Calculate date and time values

DATA step functions that calculate differences between *date* and *datetime* values; or increment date and datetime values to create new date and datetime values.

INTCK 🔗	709
Returns a count of completed intervals between the supplied <i>start-date</i> and <i>end-date</i> .	
INTNX 🔗	713
Returns a numeric time, date or datetime value calculated as an <i>increment</i> number of <i>intervals</i> , added to the <i>start-date</i> .	
DATDIF 🔗	720
Returns the number of days difference between two dates as a numeric value.	
YRDIF 🔗	721
Returns the number of years difference between two dates as a numeric value.	

INTCK

Returns a count of completed intervals between the supplied *start-date* and *end-date*.



Date, datetime, and time values can be used with this function. The accuracy of the count returned reflects the combination of interval label and date, datetime or time value:

- If working with date formats, intervals covering year to day will return correct results. Do not attempt to use `HOUR`, `MINUTE`, or `SECOND` intervals with date values.
- If working with datetime formats, use increment names starting with `DT`. All values from `DTYEAR` to `DTSECOND` can be used. In addition, `HOUR`, `MINUTE` and `SECOND` intervals can be used with datetime values.
- If working with time formats, use the `HOUR`, `MINUTE` and `SECOND` intervals.

Return type: Numeric

interval

"YEAR"

Counts the number of years between the specified *start-date* and *end-date*. To use the year interval with a date value, specify either `YEAR` or `YR`; to use this interval with a datetime value, specify either `DTYEAR` or `DTYR`.

Using the `DTYEAR` interval with a date value results in an incorrect count.

"SEMIYEAR"

Counts the number of semi-years (six-month periods) between the specified *start-date* and *end-date*. To use the semiyear interval with a date value, specify either `SEMIYEAR` or `SEMI`; to use this interval with a datetime value, specify either `DTSEMIYEAR` or `DTSEMIYR`.

Using the `DTSEMIYEAR` interval with a date value results in an incorrect count.

"QUARTER"

Counts the number of quarters (three-month periods) between the specified *start-date* and *end-date*. To use the quarter interval with a date value, specify either `QUARTER` or `QTR`; to use this interval with a datetime value, specify either `DTQUARTER` or `DTQTR`.

Using the `DTQUARTER` interval with a date value results in an incorrect count.

"MONTH"

Counts the number of months between the specified *start-date* and *end-date*. To use the month interval with a date value, specify either `MONTH` or `MON`; to use this interval with a datetime value, specify either `DTMONTH` or `DTMON`.

Using the `DTMONTH` interval with a date value results in an incorrect count.

"SEMIMONTH"

Counts the number of semi-months (two-week periods) between the specified *start-date* and *end-date*. To use the semimonth interval with a date value, specify either `SEMIMONTH` or `SEMIMON`; to use this interval with a datetime value, specify either `DTSEMIMONTH` or `DTSEMIMON`.

Using the `DTSEMIMONTH` interval with a date value results in an incorrect count.

"TENDAY"

Counts the number of tenday periods between the specified *start-date* and *end-date*. To use this interval with a date value, specify `TENDAY`; to use this interval with a datetime value, specify `DTTENDAY`.

Using the `DTTENDAY` interval with a date value results in an incorrect count.

"WEEK"

Counts the number of weeks between the specified *start-date* and *end-date*. To use this interval with a date value, specify `WEEK`; to use this interval with a datetime value, specify `DTWEEK`. When using a discrete method, the default first day of a week is Sunday.

Using the `DTWEEK` interval with a date value results in an incorrect count.

"WEEKDAY"

Counts the number of weekdays (ignoring Saturdays and Sundays) between the specified *start-date* and *end-date*. To use this interval with a date value, specify `WEEKDAY`; to use this interval with a datetime value, specify `DTWEEKDAY`.

"DAY"

Counts the number of days between the specified *start-date* and *end-date*. To use this interval with a date value, specify `DAY`; to use this interval with a datetime value, specify `DTDAY`.

Using the `DTDAY` interval with a date value results in an incorrect count.

"HOUR"

This interval can only be used with datetime or time values.

Counts the number of number of hours between the datetime or time values specified in *start-date* and *end-date*. To use this interval specify either `HOUR` or `DTHOUR`.

"MINUTE"

This interval can only be used with datetime or time values.

Counts the number of number of minutes between the datetime or time values specified in *start-date* and *end-date*. To use this interval specify one of `MINUTE`, `MIN`, `DTMINUTE`, or `DTMIN`. The calculated output is the same whichever alias is selected.

"SECOND"

This interval can only be used with datetime or time values.

Counts the number of number of seconds between the datetime or time values specified in *start-date* and *end-date*. To use this interval specify one of `SECOND`, `SEC`, `DTSECOND`, or `DTSEC`. The calculated output is the same whichever alias is selected.

start-date

Type: Numeric

The start date, datetime, or time value from which the number of increments is counted. This can be a known value, a literal, or calculated using other functions. The *start-date* is not included in the final calculation.

end-date

Type: Numeric

The end date, datetime, or time value to which the number of increments is counted. This can be a known value, a literal, or calculated using other functions.

method

Optional argument

"DISCRETE"

Counts the number of natural boundaries for the interval type between the *start-date* and *end-date*. Counting natural boundaries is the default behaviour, and can lead to unexpected results. For example:

```
DATA _NULL_;  
  V = INTCK("YEAR", "31-DEC-2015"D, "01-JAN-2017"D);  
  PUT V;  
RUN;
```

Returns a count of two years, as there are two natural year boundaries between the dates, even though the dates are one year and one day apart.

"DISC"

Counts the number of natural boundaries for the interval type between the *start-date* and *end-date*. "DISC" is an alias of "DISCRETE".

"D"

Counts the number of natural boundaries for the interval type between the *start-date* and *end-date*. "D" is an alias of "DISCRETE".

"CONTINUOUS"

Counts the completed number of increments between the *start-date* and *end-date*, without reference to the natural boundary of the increment type.

"CONT"

Counts the completed number of increments between the *start-date* and *end-date*, without reference to the natural boundary of the increment type. "CONT" is an alias of "CONTINUOUS".

"C"

Counts the completed number of increments between the *start-date* and *end-date*, without reference to the natural boundary of the increment type. "C" is an alias of "CONTINUOUS".

Example

The following uses `INTCK` with a continuous method to find the current age in years, assuming a birth date of January 09 1982.

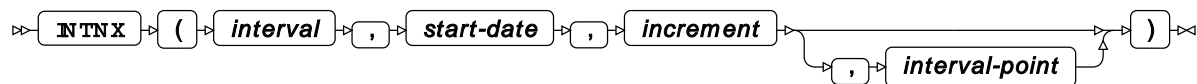
```
DATA _NULL_;
  birthDate   = "09-JAN-1982"D;
  currentDate = TODAY();
  cAge = INTCK("YEAR", birthDate, currentDate, "CONTINUOUS");
  PUT "Current age: "cAge;
RUN;
```

Which produces the following output in the log:

```
Current age: 34
```

INTNX

Returns a numeric time, date or datetime value calculated as an *increment* number of *intervals*, added to the *start-date*.



The calculated value takes account of the *interval-point* before the value is returned. The format of the value you require (date or datetime) from the function determines the interval label:

- For date values, increments covering year to day will return correct results. Do not attempt to use `HOUR`, `MINUTE`, or `SECOND` increments with date values.
- For datetime values, use increment names starting with `DT`. All values from `DTYEAR` to `DTSECOND` can be used. In addition, `HOUR`, `MINUTE` and `SECOND` increments can be used with datetime values.
- For time values, use the `HOUR`, `MINUTE` and `SECOND` increments.

Return type: Numeric

interval

The "units" by which *start-date* is incremented. Once the date has been calculated, the return value is determined by the interval-point selected, and varies dependent on the *interval* type selected.

"YEAR"

This *interval* increments a date or datetime by one year at a time. To use the year *interval* with a date value, specify either `YEAR` or `YR`; to use this *interval* with a datetime value, specify either `DTYEAR` or `DTYR`.

The interval-point selected may have a notable effect on the accuracy of the result. Once the calculation is complete, the result is rounded to return January 1 of the year for an interval-point of `BEGINNING`; July 1 or July 2 in leap years for an interval-point of `MIDDLE`; and December 31 for an interval-point of `END`. An interval-point of `SAME` preserves the offset from the date or datetime within the resultant *interval* period.

Using the `DTYEAR` *interval* with a date value results in an inaccurate result.

"SEMIYEAR"

This *interval* increments a date or datetime by six months at a time. To use the semiyear *interval* with a date value, specify either `SEMIYEAR` or `SEMIYR`; to use this *interval* with a datetime value, specify either `DTSEMIYEAR` or `DTSEMIYR`.

The interval-point selected may have a notable effect on the accuracy of the result. An interval-point of `SAME` preserves the offset from the date or datetime within the resultant *interval* period; for other interval-points once the calculation is complete the result is rounded and, depending on which half of the year, the date or datetime value:

- For the first half of the year: January 1 for an interval-point of `BEGINNING`; April 1 for an interval-point of `MIDDLE`; and June 30 for an interval-point of `END`.
- For the second half of the year: July 1 of the year for an interval-point of `BEGINNING`; September 30 for an interval-point of `MIDDLE`; and December 31 for an interval-point of `END`.

"QUARTER"

This *interval* increments a date or datetime by 3 months at a time. To use the quarter *interval* with a date value, specify either `QUARTER` or `QTR`; to use this *interval* with a datetime value, specify either `DTQUARTER` or `DTQTR`.

The interval-point selected may have an effect on the result. An interval-point of `SAME` preserves the offset from the date or datetime within the resultant *interval* period; for other interval-points once the calculation is complete the result is rounded and, depending on which quarter of the year, the date or datetime value:

- For the first quarter of the year: January 1 for an interval-point of `BEGINNING`; February 14 (February 15 in leap years) for an interval-point of `MIDDLE`; and March 31 for an interval-point of `END`.
- For the second quarter of the year: April 1 for an interval-point of `BEGINNING`; May 16 for an interval-point of `MIDDLE`; and June 30 for an interval-point of `END`.
- For the third quarter of the year: July 1 for an interval-point of `BEGINNING`; August 15 for an interval-point of `MIDDLE`; and September 30 for an interval-point of `END`.

- For the fourth quarter of the year: October 1 for an interval-point of `BEGINNING`; November 15 for an interval-point of `MIDDLE`; and December 31 for an interval-point of `END`.

"MONTH"

This *interval* increments a date or datetime by one month at a time. To use the month *interval* with a date value, specify either `MONTH` or `MON`; to use this *interval* with a datetime value, specify either `DTMONTH` or `DTMON`.

The interval-point selected may have an effect on the result. An interval-point of `SAME` preserves the offset from the date or datetime within the resultant *interval* period; for other interval-points once the calculation is complete the result is rounded, and the result returned depends on the interval-point selected:

- With an interval-point of `BEGINNING`, using date values, returns the first day of the month containing the unrounded result; for datetime values, returns a value corresponding to midnight for the first day of the month containing the unrounded result.
- With an interval-point of `MIDDLE`, using date values, returns the middle day of the month containing the unrounded result; for datetime values, returns a value corresponding to 11:59:59 of the middle day of the month containing the unrounded result.
- With an interval-point of `END`, using date values, returns the last day of the month containing the unrounded result; for datetime values, returns a value corresponding to 23:59:59 of the last day of the month containing the unrounded result.

"SEMIMONTH"

This *interval* increments a date or datetime by half a month at a time. A semimonth is fixed in the SAS language to: the first if the month to the fifteenth being the first half; the sixteenth of the month to the final day as the second half. To use the semimonth *interval* with a date value, specify either `SEMIMONTH` or `SEMIMON`; to use this *interval* with a datetime value, specify either `DTSEMIMONTH` or `DTSEMIMON`.

The interval-point selected may have an effect on the result. An interval-point of `SAME` preserves the offset from the date or datetime within the resultant *interval* period; for other interval-points once the calculation is complete the result is rounded depending on the interval-point selected:

- With an interval-point of `BEGINNING`. For the first half of the month, returns the first of the month for date values, and midnight on the first of the month for datetime values. For the second half of the month returns the sixteenth for date values, and midnight on the sixteenth of the month for datetime values.
- With an interval-point of `MIDDLE`. For the first half of the month, returns the eighth of the month for date values, and 11:59:59 on the eighth of the month for datetime values. For the second half of the month returns the middle-day between the sixteenth and the month end for date values, and 23:59:59 on the same day for datetime values.

- With an interval-point of `END`. For the first half of the month, returns the fifteenth of the month for date values, and 11:59:59 on the fifteenth for datetime values. For the second half of the month returns the last day of the month for date values, and 23:59:59 on the last day of the month for datetime values.

"TENDAY"

The *tenday interval* splits a month into ten day intervals, starting on the first, eleventh and twenty-first days of the month. Where a month has 31 days, the third *tenday interval* includes the thirty-first of the month.

This *interval* increments a date or datetime by one tenday period at a time. To use this *interval* with a date value, specify `TENDAY`; to use this *interval* with a datetime value, specify `DTTENDAY`.

Using an interval-point of `SAME`, the offset is maintained within the resultant interval, including the time portion of a datetime value. When using any other interval-point, once the calculation is complete the result is rounded depending on the interval-point selected:

- With an interval-point of `BEGINNING`. For date values the first *interval* in a month returns the first of the month; the second interval returns the eleventh of the month; the third *interval* returns the twenty-first of the month. Datetime values return midnight on the same days in the intervals.
- With an interval-point of `MIDDLE`. For date values the first *interval* in a month returns the fifth of the month; the second interval returns the fifteenth of the month; the third *interval* returns either the twenty-fifth if the month has thirty day, or the twenty-sixth if the month has thirty-one days. Datetime values return midnight on the same days in the intervals.
- With an interval-point of `END`. For the first half of the month, returns the fifteenth of the month for date values, and 11:59:59 on the fifteenth for datetime values. For the second half of the month returns the last day of the month for date values, and 23:59:59 on the last day of the month for datetime values.

"WEEK"

This *interval* increments a date or datetime by one week at a time. When using a week increment, the default first day of a week is Sunday.

Using an interval-point of `SAME`, the offset is maintained within the resultant interval, including the time portion of a datetime value. When using any other interval-point, once the calculation is complete the result is rounded depending on the interval-point selected:

- An interval-point of `BEGINNING` returns the default first day of a week. Date values with return the date for the Sunday of the resultant interval; datetime values return midnight on Sunday of the resultant interval.
- With an interval-point of `MIDDLE`. Date values return the date for the Wednesday of the resulting interval, datetime values return 11:59:59 on the Wednesday of the resultant interval.

- An interval-point of `END` returns the default final day of a week. Date values return the date for the Saturday of the resulting interval; datetime values return 23:59:59 on the Saturday of the resultant interval.

"WEEKDAY"

This *interval* increments a date or datetime by one day at a time, ignoring Saturdays and Sundays. Adding one weekday to a Friday therefore returns the date for the following Monday, adding two days returns the date for the following Tuesday.

To use this *interval* with a date value, specify `WEEKDAY`; to use this *interval* with a datetime value, specify `DTWEEKDAY`. Using the `DTWEEKDAY` *interval* with a date format results in an inaccurate date, as the calculated result is rounded to the entered interval-point in seconds before being returned from the function. Using the `WEEKDAY` *interval* with a datetime format results in a missing value.

"DAY"

This *interval* increments a date or datetime by one day at a time. To use this *interval* with a date value, specify `DAY`; to use this *interval* with a datetime value, specify `DTDAY`.

Using the `DTDAY` *interval* with a date format results in an inaccurate date, as the calculated result is rounded to the entered interval-point in seconds before being returned from the function. Using the `DAY` *interval* with a datetime format results in a missing value.

"HOUR"

This *interval* can only be used with datetime or time values.

To use the hour interval, specify one of the following as the interval: `hour`, `hr`, `dthour`, or `dthr`. The calculated output is the same whichever alias is selected.

If you apply an hour *interval* to a date value the initial date is treated as the number of seconds in a datetime value to which `increment x 3600` seconds is added. The calculated value is rounded to increment-point, and the output treated as a day count resulting in an inaccurate date.

The following example shows the output when using datetime and date formats with an *interval* denominated in hours:

```
DATA _NULL_;
  DATE1 = 1727568000;
  DATE2 = "29-SEP-2014"d;

  DOUT1 = INTNX ("HOUR", DATE1, 11);
  DOUT2 = INTNX ("HOUR", DATE2, 11);
  PUT "Starting DATETIME:           " DATE1;
  PUT "INTNX DATETIME raw data:      " DOUT1;
  PUT "DATETIME format with hours:   " DOUT1 DATETIME.;
  PUT "Starting DATE:                " DATE2;
  PUT "INTNX DATE raw data:          " DOUT2;
  PUT "DATE format with hours:       " DOUT2 DATE11.;
RUN;
```

Which produces the following output in the log. The INTNX DATE raw data has been rounded to the beginning of the interval, and must therefore be divisible by 3600.

```
Stating DATETIME:          1727568000
INTNX DATETIME raw data:   1727607600
DATETIME format with hours: 29SEP14:11:00:00
Starting DATE:             19995
INTNX DATE raw data:       57600
DATE format with hours:    14-SEP-2117
```

"MINUTE"

This *interval* should only be used with datetime or time values.

To use the minute interval, specify one of the following as the interval: MINUTE, MIN, DTMINUTE, or DTMIN. The calculated output is the same whichever alias is selected.

If you apply a minute *interval* to a date value, the initial date is treated as the number of seconds in a datetime value, to which `increment x 60 seconds` is added. The calculated value is rounded to increment-point, and the output treated as a day count resulting in an inaccurate date.

The following example shows the output when using datetime and date formats with an *interval* denominated in minutes:

```
DATA _NULL_;
  DATE1 = 1727568000;
  DATE2 = "29-SEP-2014"d;

  DOUT1 = INTNX ("MINUTE", DATE1, 1001);
  DOUT2 = INTNX ("MINUTE", DATE2, 1001);
  PUT "Stating DATETIME:          " DATE1;
  PUT "INTNX DATETIME raw data:    " DOUT1;
  PUT "DATETIME format with minutes: " DOUT1 DATETIME.;
  PUT "Starting DATE:             " DATE2;
  PUT "INTNX DATE raw data:        " DOUT2;
  PUT "DATE format with minutes:    " DOUT2 DATE11.;
RUN;
```

Which produces the following output. The INTNX DATE raw data has been rounded to the beginning of the interval, and must therefore be divisible by 60.

```
Stating DATETIME:          1727568000
INTNX DATETIME raw data:   1727628060
DATETIME format with minutes: 29SEP14:16:41:00
Starting DATE:             19995
INTNX DATE raw data:       80040
DATE format with minutes:   21-FEB-2179
```

"SECOND"

This *interval* should only be used with datetime or time values.

To use the second interval, specify one of the following as the interval: SECOND, SEC, DTSECOND, or DTSEC. The calculated output is the same whichever alias is selected.

If you apply this *interval* to a date value, the date is treated as the number of seconds in a datetime value, to which the *increment* number of seconds is added and the result converted to an inaccurate date.

The following example shows the output when using datetime and formats with an *interval* date denominated in seconds:

```
DATA _NULL_;
  DATE1 = 1727568000;
  DATE2 = "29-SEP-2014"d;

  DOUT1 = INTNX ("SECOND", DATE1, 10001);
  DOUT2 = INTNX ("SECOND", DATE2, 10001);
  PUT "Stating DATETIME:           " DATE1;
  PUT "INTNX DATETIME raw data:    " DOUT1;
  PUT "DATETIME format with seconds: " DOUT1 DATETIME.;
  PUT "Starting DATE:             " DATE2;
  PUT "INTNX DATE raw data:        " DOUT2;
  PUT "DATE format with seconds:   " DOUT2 DATE11.;
RUN;
```

Which produces the following output. Setting the *interval* to seconds means the calculated result is not rounded.

```
Stating DATETIME:           1727568000
INTNX DATETIME raw data:    1727578001
DATETIME format with seconds: 29SEP14:02:46:41
Starting DATE:             19995
INTNX DATE raw data:        29996
DATE format with seconds:   15-FEB-2042
```

start-date

Type: Numeric

The start date, time or datetime to which an increment number of intervals is added. This can be a known value, a literal, or calculated using other functions that return a date, time or datetime value.

increment

Type: Numeric

The number of the selected interval to be added to the start-date.

interval-point

Optional argument

Determines the return point within the *interval*. When not entered, the default is the start of the selected interval. The calculated value is rounded to the *interval-point* selected before being returned by *INTNX*.

"BEGINNING"

Rounds the output to the start point of the current *interval* period.

"MIDDLE"

Rounds the output to the mid-point of the current *interval* period.

"END"

Rounds the output to the end-point of the current *interval* period.

"SAME"

Preserves the offset within the *interval* period. Alternatively, you can specify the alias "SAMEDAY".

DATDIF

Returns the number of days difference between two dates as a numeric value.

➤ DATDIF ➤ (➤ *start-date* ➤ , ➤ *end-date* ➤ , ➤ *basis* ➤) ➤

The returned value is resolved using the basis, which determines how month and years are calculated between the *start-date* and *end-date*. This function can only be used with date values; datetime values can be passed in to the function, but differences cannot be calculated between datetime values.

Return type: Numeric

start-date

Type: Numeric

The date value from which to begin counting, inclusive. This can be a known date value, a date literal, or calculated using other functions that return a date value.

end-date

Type: Numeric

The date value at which to stop counting. This date is excluded from the count. This can be a known date value, a date literal, or calculated using other functions that return a date value.

basis

The basis determines how the days in the month and days in the year are calculated.

"30/360"

Each month is treated as containing 30 days, and the year 360 days. Differences within a month return the actual value; where a difference crosses the month boundary the returned value is rounded to a 30 day month. Where the difference is greater than a year, the count is rounded assuming a 360 day year.

For example:

```
DATA _NULL_;
  startDate = "01-JAN-2015"D;
  endDate1  = "10-MAR-2015"D;
  endDate2  = "20-APR-2016"D;
  outDate1 = DATDIF(startDate, endDate1, "30/360");
  outDate2 = DATDIF(startDate, endDate2, "30/360");
  PUT "count between startDate and";
  PUT "endDate1: " outDate1;
  PUT "endDate2: " outDate2;
RUN;
```

Returns the following counts. Counting to `endDate1` is two months and nine days difference, giving a total count of 69 days; `endDate2` is greater than one year in difference, and can be calculated as one year (360 days), three months (90 days) and 19 days for a total of 469 days.

```
count between startDate and
endDate1: 69
endDate2: 469
```

"360"

An alias of "30/360".

"ACT/ACT"

Returns the actual count of days between the *start-date* and *end-date*.

"ACTUAL"

An alias of "ACT/ACT", returning the actual count of days between the *start-date* and *end-date*.

"ACT/360"

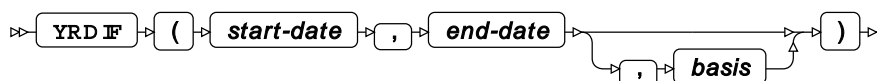
Returns the actual count of days between the *start-date* and *end-date*, but assume 360 days in the year when the difference is greater than a year.

"ACT/365"

Returns the actual count of days between the *start-date* and *end-date*, but assume 365 days in the year when the difference is greater than a year.

YRDIF

Returns the number of years difference between two dates as a numeric value.



Return type: Numeric

start-date

Type: Numeric

The date value from which to begin counting, inclusive. This can be a known date value, a date literal, or calculated using other functions that return a date value.

end-date

Type: Numeric

The date value at which to stop counting. This date is excluded from the count. This can be a known date value, a date literal, or calculated using other functions that return a date value.

basis

Optional argument

The basis determines how the days in the year are calculated, and consequently the difference in years between two dates.

"AGE"

An alias of "ACTUAL".

"30/360"

Returns the number of years' difference between the *start-date* and *end-date*, assuming that each year contains 360 days. Where the result contains a fraction of a year, the actual days within a month are returned, where the count crosses a month boundary, a 30 day month is assumed, and the count is rounded before being returned as the fractional part of the result.

"ACT/ACT"

An alias of "ACTUAL".

"ACTUAL"

Returns the number of years' difference between the *start-date* and *end-date*, taking into account the actual number of days in each year.

"ACT/360"

Returns the number of years' difference between the *start-date* and *end-date*, assuming that each year contains 360 days. Where the result contains a fraction of a year, the actual days elapsed within the year is returned as the fractional part of the result.

"360"

Returns the number of years' difference between the *start-date* and *end-date*, assuming that each year contains 360 days. The *start-date* and *end-date* are converted to date formats and the difference is calculated as the number of elapsed days divided by 360. This calculation does not take into account the extra five days in each year, and may generate some unexpected results, for example:

```
DATA _NULL_;
  start="01-JAN-2000"D;
  current = "01-JAN-2016"D;
  C_360    = YRDIF(start, CURRENT, "360");
  PUT C_360;
RUN;
```

Where C_360 returns a value of 16.233333, rather than a value of 16 that might have been expected as both dates are January 1.

"ACT/365"

Returns the number of years' difference between the *start-date* and *end-date*, assuming that each year contains 365 days. The *start-date* and *end-date* are converted to date formats and the difference is calculated as the number of elapsed days divided by 365. Where the result contains a fraction of a year, the actual days elapsed within the year is returned as the fractional part of the result.

"365"

Returns the number of years' difference between the *start-date* and *end-date*, assuming that each year contains 365 days. The *start-date* and *end-date* are converted to date formats and the difference is calculated as the number of elapsed days divided by 365.

Dataset input and output functions and CALL routines

Open and close datasets, get information about datasets, and get observations from them.

An identifier is returned when you open a dataset. This identifier can then be used with the other input and output functions to return information about the attributes of the dataset, the format of variables in the dataset, the library name and reference for the dataset, and so on. You can also get observations from the dataset and variables from a specified observation.

ATTRC [↗](#).....725
Returns information about character-based attributes of a dataset, such as its character set, whether it is encrypted, the library with which it is associated, and so on.

ATTRN ↗	727
Returns information about the numeric attributes of a dataset, such as the number of observations, the date and time the dataset was created and modified, and so on.	
CEXIST ↗	731
Returns a value that indicates whether a specified catalog or catalog entry exists.	
CLOSE ↗	731
Closes a dataset opened with the <code>OPEN</code> function. The dataset to be closed is specified using the dataset identifier returned by <code>OPEN</code> .	
CUROBS ↗	732
Returns the position of the current observation in the dataset. For example, if the <code>FETCHOBS</code> function has been used to get the tenth observation in the current dataset, then <code>CUROBS</code> will return 10.	
DESCRIBE ↗	733
Enables a list to be populated with dataset, data view and catalog entry attributes.	
DSNAME ↗	737
Returns the name of an open dataset associated with a specified identifier.	
EXIST ↗	738
Returns a value that specifies whether a specified member exists in a library. You can also specify the type of member, if required.	
FETCH ↗	739
Fetches the next observation from a dataset.	
FETCHOBS ↗	742
Fetches the specified observation from a dataset.	
GETVARC ↗	744
Returns the value of the specified character variable from an observation in the Dataset Data Vector (DDV).	
GETVARN ↗	746
Returns the value of the specified numeric variable from the current observation in the Dataset Data Vector (DDV).	
IORCMMSG ↗	747
Returns an error message.	
LIBNAME ↗	747
Assigns or deassigns a libref for a library.	
LIBREF ↗	750
Returns a value indicating whether a specified libname has been assigned.	
NOTE ↗	751
Returns an identifier for the current observation from a specified open dataset.	
OPEN ↗	752
Opens a specified dataset, and returns an identifier that can be used by other functions.	
POINT ↗	753
Points at an observation previously identified by the <code>NOTE</code> function.	

REWIND ↗	754
Positions the dataset pointer at the first record in the dataset.	
VARFMT ↗	755
Returns the format of a specified variable, if a format has been applied.	
VARINFMT ↗	756
Returns the informat of a specified variable, if an informat has been applied.	
VARLABEL ↗	758
Returns the label associated with the specified variable.	
VARLEN ↗	759
Returns the length of a specified variable.	
VARNAME ↗	760
Returns the name of a specified variable. The name is that defined for the variable in the dataset.	
VARNUM ↗	761
Returns the ordinal position of a named variable in an observation in a dataset.	
VARTYPE ↗	762
Returns the type of the variable at the specified ordinal position in an observation. A variable can be numeric or character.	
CALL SET ↗	763
Identifies the variables in a dataset, and makes them available to other functions in the <code>DATA</code> step. This enables the variable names specified in the dataset to be used in other functions and routines.	

ATTRC

Returns information about character-based attributes of a dataset, such as its character set, whether it is encrypted, the library with which it is associated, and so on.

```

>> ATTRC ( dataset-id , attribute )

```

For information on the numeric attributes of datasets, see [ATTRN](#) [↗](#) (page 727).

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset for which you want information. The identifier is returned by the [OPEN](#) [↗](#) (page 752) function, which should be called before this function is used.

attribute

Specifies the attribute for which information will be returned. The attributes listed below are available. The value returned by an attribute is also described.

"CHARSET"

The character set in which the data is stored.

"ENCRYPT"

A value indicating whether the data is encrypted or not. YES if it is, otherwise NO.

"ENGINE"

The name of the engine used to open the file. For example, WPD for WPS's own dataset files.

"LABEL"

The label associated with the dataset.

"LIB"

The name of the library assigned to a dataset (if one has been assigned).

"MEM"

The name of the dataset in use.

"MODE"

The mode with which the file was opened. This will be I, IN or IS. See [OPEN](#) (page 752) for details.

"MTYPE"

A value indicating whether the dataset is a view (VIEW) or data (DATA).

"SORTEDBY"

If the dataset is sorted, the observation on which the data is sorted. Otherwise, null.

"SORTLVL"

A value indicating that the dataset has been sorted, and the type of sort. If the dataset has not been sorted, the value is null, otherwise, the value can be:

- **WEAK** – The dataset has been defined as sorted using the SORTEDBY option to the DATA statement, in which case the sort order has not been validated by WPS, and is shown as not validated in the metadata for the dataset.
- **STRONG** – The dataset has been sorted using mechanisms such as PROC SORT, in which case the sort order is validated by WPS, and is shown as validated in the metadata for the dataset.

"SORTSEQ"

The dataset collation order, such as ASCII or EBCDIC, or one of the national collation ordering schemes. The collation order is set using the `SORTSEQ` system option or the `PROC SORT` option `SORTSEQ`. If the collating sequence is the same as the device on which the `DATA` step is run, then null is returned.

"TYPE"

Returns the value specified for the `DATA` step option `TYPE=`. If no value has been set for the option, blank is returned.

Example

In this example, the function is used to return various attributes of the dataset. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');

  ca = ATTRC(id,"ENCRYPT");
  PUT 'Is the dataset encrypted? ' ca;

  ca = ATTRC(id,"ENGINE");
  PUT 'Which engine was used to open the dataset? ' ca;

  ca = ATTRC(id,"CHARSET");
  PUT 'Which character set does the dataset use? ' ca;

  ca = ATTRC(id,"MODE");
  PUT 'In which mode was the dataset opened? ' ca;

  returnc = CLOSE(id);
RUN;
```

This produces the following output:

```
Is the dataset encrypted? NO
Which engine was used to open the dataset? WPD
Which character set does the dataset use? ASCII
In which mode was the dataset opened? I
```

ATTRN

Returns information about the numeric attributes of a dataset, such as the number of observations, the date and time the dataset was created and modified, and so on.

➤ **ATTRN** ➤ (➤ *dataset-id* ➤ , ➤ *attribute* ➤) ➤

For information on the character attributes of datasets, see [ATTRC](#) (page 725).

Return type: Numeric

dataset-id

Type: Numeric

The identifier of the dataset for which you want information. The identifier is returned by the `OPEN` [↗](#) (page 752) function, which should be called before this function is used.

attribute

Specifies the attribute for which information will be returned. The attributes listed below are available. The value returned by an attribute is also described.

"ANOBS"

Indicates whether the number of observations in the dataset are known; 1 if true, 0 otherwise.

"ANY"

Indicates whether variables are identified or not. If no variables are identified, -1 is returned. If variables are identified, but there are no observations, 0 is returned. Otherwise, 0 is returned.

"ARAND"

Indicates whether random access is allowed to the dataset; 1 if true, 0 otherwise.

"ARWU"

Indicates whether the dataset is read only; 1 if true, 0 otherwise.

"AUDIT"

Indicates whether audit logging is supported; 1 if true, 0 otherwise.

Note:

Audit logging is not currently supported

"AUDIT_DATA"

Indicates whether after images of updated records are stored; 1 if true, 0 otherwise.

Note:

Audit logging is not currently supported

"AUDIT_BEFORE"

Indicates whether before images of updated records are stored; 1 if true, 0 otherwise.

Note:

Audit logging is not currently supported

"AUDIT_ERROR"

Indicates whether an attempt to store after images of updated records have been unsuccessful; 1 if true, 0 otherwise.

Note:

Audit logging is not currently supported

"CRDTE"

Date and time the dataset was created.

"ICONST"

A value identifying the type of integrity constraint.

Note:

Integrity constraints are not currently supported

"INDEX"

Indicates whether the dataset supports indexing; 1 if true, 0 otherwise.

"ISINDEX"

Indicates whether the dataset is indexed; 1 if true, 0 otherwise.

"ISSUBSET"

Indicates the observations in the dataset are a subset of a table; 1 if true, 0 otherwise.

"LRECL"

The logical record length of the dataset.

"MODTE"

The date and time at which the dataset was last modified.

"NDEL"

The number of observations marked for deletion.

"NLOBS"

The number of observations not marked for deletion.

"NLOBSF"

The number of observations in the dataset, if known; -1 otherwise. For example, if the dataset is an SQL view or a dataset view, the number of observations contained cannot be known beforehand, so for such a view -1 would be returned.

"NOBS"

The number of observations, including deleted observations.

"NVAR"

The number of variables in the dataset.

"RANDOM"

Indicates whether the dataset can be accessed randomly; 1 if true, 0 otherwise.

"TAPE"

Indicates whether the dataset can be accessed sequentially; 1 if true, 0 otherwise.

"VAROBS"

Indicates whether the number of observations in the dataset is known; 1 if true, 0 otherwise.

"WHSTMT"

Indicates whether a dataset `WHERE` statement is active, and if so, what type. The value will be 0, if no `WHERE` statement is active, or 2 if a temporary `WHERE` clause is active.

Example

In this example, the function is used to return various numeric attributes of the dataset. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');

  na = ATTRN(id,"LRECL");
  PUT 'The logical record length of the dataset is: ' na;

  na = ATTRN(id,"ISINDEX");
  PUT 'Is the dataset index sequential? ' na;

  nd = ATTRN(id,"CRDTE");
  PUT 'The dataset was last modified on ' nd datetime.;

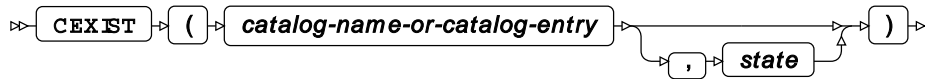
  returnc = CLOSE(id);
RUN;
```

This produces the following output:

```
The logical record length of the dataset is: 284
Is the dataset index sequential? 1
The dataset was last modified on 01FEB16:07:00:01
```


CEXIST

Returns a value that indicates whether a specified catalog or catalog entry exists.



Return type: Numeric

Returns 1 (true) if the catalog or catalog entry exists, 0 otherwise).

catalog-name-or-catalog-entry

Type: Character

The catalog name or catalog entry.

state

Optional argument

Only one value is available:

"U"

Checks whether the catalog or catalog entry is also updatable, that is, it has write access.

Example

In this example, it is assumed the catalog specified exists. The result is written to the log.

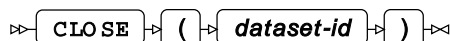
```
DATA _NULL_;
  FILENAME F CATALOG 'work.cat1.entry.demo';
  result = CEXIST('work.cat1');
  op = IFC(result,"The catalog entry exists","The catalog entry doesn't exist");
  PUT op;
RUN;
```

This produces the following output:

```
The catalog entry exists
```

CLOSE

Closes a dataset opened with the `OPEN` function. The dataset to be closed is specified using the dataset identifier returned by `OPEN`.



Return type: Numeric

If the dataset is successfully closed, 0 (zero) is returned; otherwise 1 is returned.

dataset-id

Type: Numeric

The identifier of the dataset to close. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

`OPEN` returns 0 (zero) if the file specified does not exist. `CLOSE` does not recognise 0 as valid identifier, and will return an error if you attempt to close a file using it.

Example

In this example, the identifier returned by the `OPEN` function is used in the subsequent `CLOSE` function. The result is written to the log.

```
DATA _NULL_;  
  id = OPEN('sashelp.zipcode');  
  PUT "The identifier is: " id;  
  id=1;  
  returnc = CLOSE(id);  
  IF returnc = 0 THEN cs = 'Dataset closed successfully';  
    ELSE cs = 'Dataset not closed';  
  PUT cs;  
RUN;
```

This produces the following output:

```
The identifier is: 1  
Dataset closed successfully
```

CUROBS

Returns the position of the current observation in the dataset. For example, if the `FETCHOBS` function has been used to get the tenth observation in the current dataset, then `CUROBS` will return 10.

» `CUROBS` (*dataset-id*) «

The current observation is stored in the Dataset Data Vector (DDV).

Return type: Numeric

dataset-id

Type: Numeric

The identifier of the dataset that contains the current observation for which you want the position. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

Example

In this example, the function returns the position of the observation selected by the `FETCHOBS` function. The result is written to the log.

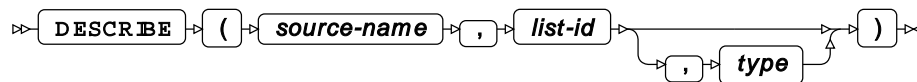
```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  fo = FETCHOBS(id,10);
  return = CUROBS(id);
  cds = CLOSE(id);
  PUT 'The current observation is: ' return;
RUN;
```

This produces the following output:

```
The current observation is: 10
```

DESCRIBE

Enables a list to be populated with dataset, data view and catalog entry attributes.



This function requires a list to be created using one of the list creation functions, [MAKELIST](#) (page 1667) or [MAKENLIST](#) (page 1668). For datasets and dataset views, the list can then be populated with the dataset attributes returned by [ATTRC](#) (page 725) and [ATTRN](#) (page 727). For catalog entries the list can then be populated with attributes returned for the keywords described below.

Return type: Numeric

0 (zero) if successful.

source-name

Type: Character

The name of the dataset, data view or catalog entry for which you want information about attributes. The name must be specified using libname format, for example `lbooks.books`.

list-id

Type: List

The identifier of a list into which the attributes will be copied. The list must have item names that match the attributes to be returned. For example, if you want to return information about whether the dataset is indexed, the list must contain an item named `MODE`. See the section below for more information, and the examples.

type

Optional argument

Type: Character

The type of dataset. This can be:

'DATA'

A dataset.

'CATALOG'

A catalog entry.

'VIEW'

A data view.

If you do not specify a value for this argument, the type of dataset is determined by the number of filename elements that constitute *source-name*. For example, if you specify:

- `books` or `booklib.books`, the source is assumed to be a dataset or data view (in the first case, `work.books` is assumed).
- `booklib.books.history.catams`, the source is assumed to be a catalog entry. If you omit the last element of the name for a catalog entry (for example, `booklib.books.history`), the default is `program`.

Lists must be named lists. For datasets and data views, a list item name must be an attribute corresponding to one of the keywords listed under the attribute option of the `ATTRC` [↗](#) (page 725) and `ATTRN` [↗](#) (page 727) functions. For example, `LIB` returns as a string the library in which the dataset resides; `ANOB`s returns the value 1 or 0, depending on whether the number of observations is available.

`ATTRC` returns characters, and `ATTRN` returns a number, so the list item must be set with the appropriate function (`SETNITEMC` [↗](#) (page 1712) or `SETNITEMN` [↗](#) (page 1725)). See the examples below.

When you create a list item it can have any value, as the initial values will be replaced by the value of the corresponding attribute.

For catalog entries, a list item name can be one of the following:

'DESC'

The description provided for the catalog entry.

'EDESC'

The extended description provided for the catalog entry. Extended descriptions are not currently supported, so no value is returned.

'CRDATE'

The date at which the dataset entry was created.

'DATE'

The date at which the dataset entry was last modified.

'CRDATE' and 'DATE' can return the date as a string or as a number. If a string, the format is *dd/mmm/yy*, where *dd* is the numeric day in the month (for example, 11), *mmm* is an abbreviation for the English language name for the month (for example, SEP), and *yy* is the last two digits of the year (for example, 17). If a number, it is the number of days since epoch.

Basic example

In this example, attributes are returned from a specified dataset to a list. The resulting list is then written to the log using `CALL PUTLIST` [🔗](#) (page 1737).

```
LIBNAME BOOK_DIR 'c:\temp\books';
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'LIB');
  lr1 = SETNITEMC(lid1, '', 'MODE');
  lr1 = SETNITEMN(lid1, ., 'LRECL');

  dsc = DESCRIBE('book_dir.books', lid1);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(LIB='BOOK_DIR'
MODE='I'
LRECL=196
) [1]
```

The values of the attributes, 'LIB', 'MODE' and 'LRECL' are inserted into the list. Although the argument 'DATA' has not been specified, the function recognises that a dataset has been specified because the argument to *source-name* comprises a libname and a dataset name from the specified library.

Example – getting attribute information for a data view

In this example, attributes are returned from a specified data view into a list. The resulting list is then written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
LIBNAME BOOK_DIR 'c:\temp\books';
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'LIB');
  lr1 = SETNITEMC(lid1, '', 'MODE');
  lr1 = SETNITEMN(lid1, ., 'ANOBS');
  lr1 = SETNITEMN(lid1, ., 'INDEX');
  lr1 = SETNITEMN(lid1, ., 'LRECL');
  lr1 = SETNITEMN(lid1, ., 'ISSUBSET');

  dsc = DESCRIBE('book_dir.histbooks', lid1,'view');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(LIB='BOOK_DIR'
MODE='I'
ANOBS=1
INDEX=1
LRECL=196
CRDTE=1816773917.3
) [1]
```

The values of the attributes 'LIB', 'MODE', 'ANOBS', 'INDEX', 'LRECL', and 'CRDTE' are inserted into the list. In this case, the data is contained in a data view because *type* is specified as 'VIEW'.

Example – getting attribute information for a catalog entry

In this example, attributes are returned from a specified catalog entry into a list. The resulting list is then written to the log using `CALL PUTLIST` [\(page 1737\)](#).

```
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'DESC');
  lr1 = SETNITEMC(lid1, '', 'EDESC');
  lr1 = SETNITEMC(lid1, , 'CRDATE');
  lr1 = SETNITEMN(lid1, ., 'DATE');

  dsc = DESCRIBE('book_dir.bookscat.history1.CATAMS', lid1);

  CALL PUTLIST(lid1,,0);

  gi = GETNITEMN(lid1, 'date');
  PUT gi = date.;

RUN;
```

This produces the following output:

```
(DESC='History subset'
 EDESC=' '
 CRDATE='09/11/2017'
 DATE=21073
) [1]
Converted date 11SEP17
```

The value for 'CRDATE' has been returned as a string, while the value for 'DATE' has been returned as the number of days since epoch. In this example, the value for 'DATE' is obtained from the list and written to the log using the `DATE.` format to show that the values of 'CRDATE' and 'DATE' are the same. The filename is recognised as a catalog entry as it comprises four elements.

DSNAME

Returns the name of an open dataset associated with a specified identifier.

```
DSNAME ( dataset-id )
```

Datasets are opened using the `OPEN` function, which returns a dataset identifier that can be used in other dataset input/output functions. `DSNAME` enables you to obtain the name of the dataset associated with an identifier.

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset of interest. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

Example

In this example, the function returns the name of the dataset previously opened using the `OPEN` function. The result is written to the log.

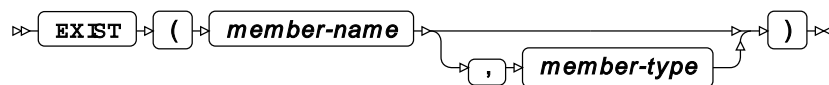
```
DATA _NULL_;  
  id = OPEN('sashelp.zipcode');  
  name = DSNAME(id);  
  cds = CLOSE(id);  
  PUT 'The current dataset is: ' name;  
RUN;
```

This produces the following output:

```
The current dataset is: SASHELP.zipcode.DATA
```

EXIST

Returns a value that specifies whether a specified member exists in a library. You can also specify the type of member, if required.



Return type: Numeric

Returns 1 if the specified member exists, 0 otherwise.

member-name

Type: Character

The name of the library member.

member-type

Optional argument

Type: Character

The type of library member. This can be, for example, `DATA` (the default) for a dataset, `CATALOG` for a catalog, and so on.

Example

In this example, the function checks for the existence of a dataset and of a catalog in a library on the Windows operating system. The result is written to the log.

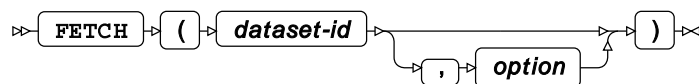
```
DATA _NULL_;  
  IF EXIST('sashelp.locale', 'catalog') THEN PUT "The library member exists";  
  ELSE PUT "The dataset doesn't exist";  
  IF EXIST('sashelp.zipcode') then PUT "The library member exists";  
  ELSE PUT "The dataset doesn't exist";  
RUN;
```

This produces the following output:

```
The library member exists  
The library member exists
```

FETCH

Fetches the next observation from a dataset.



The dataset is specified using an identifier that was generated by the `OPEN` [↗](#) (page 752) function used to open the dataset. The observation is read into the Dataset Data Vector (DDV); the observation can then be accessed by subsequent functions. By default, the observation fetched is the first observation; the next invocation of the function returns the second observation, and so on. However, if the `FETCHOBS` [↗](#) (page 742) function is first invoked with a specified observation, a subsequent `FETCH` obtains the next observation.

Return type: Numeric

Can be:

0

The observation was found and read into the DDV.

-1

The observation specified is beyond the end of the dataset

dataset-id

Type: Numeric

The identifier of the dataset from which the observation will be fetched. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

option

Optional argument

There is one optional argument:

"NOSET"

Ensures that variables in the DDV are not updated by corresponding variables obtained from the dataset by the `CALL SET` routine.

Note:

This keyword only effects the `CALL SET` routine. If you use another function (such as `GETVARN`) to obtain the variable, then `NOSET` has no effect.

If an attempt is made to fetch an observation beyond the end of the file, no observation is found, and the DDV retains the last observation successfully fetched.

Basic example

In this example, the function fetches observations that are then accessed by the `GETVARN` function, which in this example returns the third variable in the observation. The result is written to the log.

```
LIBNAME ref 'C:\program files\World Programming\WPS\4\sashelp';
DATA _NULL_;
  id = OPEN('ref.zipcode');
  DO i= 1 TO 10;
    gno = FETCH(id);
    nextrec = GETVARN(id,3);
    PUT nextrec;
  END;
  fo = FETCHOBS(id,13);
  nextrec = GETVARN(id,3);
  PUT nextrec;
  fo = FETCH(id);
  nextrec = GETVARN(id,3);
  PUT nextrec;
RUN;
```

This produces the following output:

```
-73.046388
-73.049288
-66.723627
-67.186553
-67.151954
-67.135899
-67.151346
-66.977377
-67.144161
-66.797578
-66.735892
-66.673779
```

The example output lists the third variable (in this case, a longitude) from each selected observation. The first ten observations are fetched by the `FETCH` function; each invocation of the function in the `DO` selects the next observation. The `FETCHOBS` function is then used to select a specific observation (in this example, the thirteenth in the dataset). When `FETCH` is subsequently called it gets the next observation, which in this example will be the fourteenth in the dataset.

Example – with NOSET option

In this example, the function fetches observations that are then accessed by the `GETVARN` function, and by the `CALL SET` routine. The result is written to the log.

```
DATA _NULL_;

  zip= 0.0;
  x = 0.0;
  y = 0.0;

  id = OPEN('sashelp.zipcode');

  f = FETCHOBS(id,10);
  fv = GETVARN(id,2);
  CALL SET(id);
  PUT fv= zip= x= y=;

  f = FETCH(id);
  fv = GETVARN(id,2);
  CALL SET(id);
  PUT fv= zip= x= y=;

  f = FETCH(id,"noset");
  fv = GETVARN(id,2);
  CALL SET(id);
  PUT fv= zip= x= y=;

  returnc = CLOSE(id);

RUN;
```

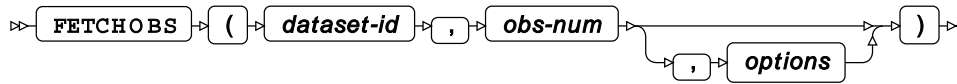
This produces the following output:

```
fv=18.287716 zip=0 x=0 y=0
fv=18.471326 zip=612 x=-66.728149 y=18.471326
fv=18.472737 zip=612 x=-66.728149 y=18.471326
```

In this example, observations are fetched from the dataset. Variables are then obtained from the observation in the DDV using `GETVARN` and `CALL SET`. The `GETVARN` function gets the second variable, which corresponds to `y` in these observations. In the second use of the `FETCH` function `NOSET` is specified. The `zip`, `x` and `y` variables remain the same as those returned in the previous `CALL SET`. However, `GETVARN` returns the value 18.456904; this demonstrates that `FETCH` did obtain the next value, and that `NOSET` only affects the `CALL SET` routine.

FETCHOBS

Fetches the specified observation from a dataset.



The dataset is specified using an identifier that was generated by the `OPEN` [\(page 752\)](#) function used to open the dataset. The observation is read into the Dataset Data Vector (DDV); the observation can then be accessed by subsequent functions.

Return type: Numeric

Can be:

0

The observation was found and read into the DDV

-1

The observation specified is beyond the end of the dataset

dataset-id

Type: Numeric

The identifier of the dataset from which the observation will be fetched. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

obs-num

Type: Numeric

The observation number of the observation to be found and then read into the DDV. The observations are numbered sequentially from one through to the end of the file.

options

Optional argument

Type: Character

NOSET

Ensures that variables in the Dataset Data Vector are not updated by corresponding variables obtained from the dataset by the `CALL SET` routine.

Note:

This keyword only effects the `CALL SET` routine. If you use another function (such as `GETVARN`) to obtain the variable, then `NOSET` has no effect.

ABS

When finding the observation specified by *obs-num*, deleted observations are by default ignored. For example, if the dataset has ten observations, two of which have been deleted, the dataset would appear to only have eight observations. If *obs-num* is set to 10, then no observation would be found and an error message returned. If this option is set, however, deleted records are not ignored and are included in the observation count. In the previous example, therefore, the last observation would now be found.

Basic example

In this example, the function fetches observations that are then accessed by the `GETVARN` function, which in this example returns the third variable in the observation. The result is written to the log.

```
LIBNAME ref 'C:\program files\World Programming\WPS\4\sashelp';
DATA _NULL_;
  id = OPEN('ref.zipcode');
  DO i= 60 to 70;
    gno = FETCHOBS(id,i);
    nextrec = GETVARN(id,3);
    PUT nextrec;
  END;
  fo = FETCHOBS(id,78);
  nextrec = GETVARN(id,3);
  PUT nextrec;
  fo = FETCH(id);
  nextrec = GETVARN(id,3);
  PUT nextrec;
RUN;
```

This produces the following output:

```
-66.614749
-66.61449
-65.732399
-66.244271
-66.39152
-65.774509
-66.015222
-66.04134
-66.036044
-66.036446
-66.614009
-66.161033
-66.161026
```

The example output lists the first variable (in this case, a longitude) from each selected observation. The first ten observations are fetched by the `FETCHOBS` function, with the index of each observation provided by the `DO` loop counter. The `FETCHOBS` function is then used to select a specific observation (in this example, observation 78 in the dataset). When `FETCH` is subsequently called, it gets the next observation, the 79th in the dataset.

Example – using NOSET option

In this example, the function fetches observations that are then accessed by the GETVARN function, and by the CALL SET routine. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');

  zip= 0.0;
  x = 0.0;
  y = 0.0;

  f = FETCHOBS(id,10);
  fv = GETVARN(id,2);
  CALL SET(id);
  PUT fv= zip= x= y=;

  f = FETCHOBS(id,11);
  fv = GETVARN(id,2);
  CALL SET(id);
  PUT fv= zip= x= y=;

  f = FETCHOBS(id,13,"noreset");
  fv = GETVARN(id,2);
  CALL SET(id);
  put fv= zip= x= y=;

  RETURN;
  CLOSE(id);

RUN;
```

This produces the following output:

```
fv=18.287716 zip=0 x=0 y=0
fv=18.471326 zip=612 x=-66.728149 y=18.471326
fv=18.456904 zip=612 x=-66.728149 y=18.471326
```

In this example, observations are fetched from the dataset. Variables are then obtained from the observation in the DDV using GETVARN and CALL SET. The GETVARN function gets the second variable, which corresponds to *y* in these observations. In the third use of the FETCHOBS function NOSET is used. The *zip*, *x* and *y* variables remain the same as those returned in the previous CALL SET. However, GETVARN returns the value 18.456904; this demonstrates that FETCH did obtain the next value, and that NOSET only affects the CALL SET routine.

GETVARC

Returns the value of the specified character variable from an observation in the Dataset Data Vector (DDV).

```
GETVARC ( dataset-id , var-num )
```

An observation can be inserted into the DDV using the `FETCH` [↗](#) (page 739) or `FETCHOBS` [↗](#) (page 742) functions. The value must be a character string, otherwise an error is returned.

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset from which to get the variable. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

var-num

Type: Numeric

The ordinal position in the observation of the variable from which you want the value. For example, if the observation has six variables, and you want to obtain the value from the fourth, you would specify 4 to this argument.

Example

In this example, the function fetches observations that are then accessed by the `GETVARC` function, which in this example returns the value of the fifth variable in the observation. The result is written to the log.

```
LIBNAME ref 'C:\program files\World Programming\WPS\4\sashelp';
DATA _NULL_;
  id = OPEN('ref.zipcode');
  DO i= 2000 TO 2010;
    gno = FETCHOBS(id, i);
    nextrec = GETVARC(id,5);
    PUT nextrec;
  END;
RUN;
```

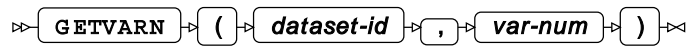
This produces the following output:

```
Derby
Derby Line
East Burke
East Charleston
East Hardwick
East Haven
East Saint Johnsbury
Glover
Granby
Greensboro
Greensboro Bend
```

The example lists the fifth variable (in this case, a city name) from a series of selected observations. Ten observations are fetched by the `FETCHOBS` function, starting at the 51st observation; each invocation of the function in the `DO` selects the next observation.

GETVARN

Returns the value of the specified numeric variable from the current observation in the Dataset Data Vector (DDV).



An observation can be inserted into the DDV using the [FETCH](#) (page 739) or [FETCHOBS](#) (page 742) functions. The variable must be numeric, otherwise an error is returned.

Return type: Numeric

dataset-id

Type: Numeric

The identifier of the dataset from which to get the variable. This is an identifier generated by the [OPEN](#) (page 752) function.

var-num

Type: Numeric

The ordinal position in the observation of the variable from which you want the value. For example, if the observation has six variables, and you want to obtain the fourth, you would specify 4 in this argument.

Example

In this example, the function fetches five observations that are then accessed by a series of `GETVARN` functions, which in this example return the first, second and third variables in the observation. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode','in');
  DO i= 192 TO 196;
    gno = FETCHOBS(id,i);
    zip = GETVARN(id,1);
    y = GETVARN(id,2);
    x = GETVARN(id,3);
    PUT "Zipcode " zip z5. " is at Y = " y 5.2 " and X = " x 7.2 ;
  END;
RUN;
```

This produces the following output:

```
Zipcode 00986 is at Y = 18.41 and X = -65.98
Zipcode 00987 is at Y = 18.41 and X = -65.98
Zipcode 00988 is at Y = 18.41 and X = -65.98
Zipcode 01001 is at Y = 42.07 and X = -72.62
Zipcode 01002 is at Y = 42.38 and X = -72.47
```


IORCMMSG

Returns an error message.



This function enables you to display a text message about the status of the input/output operation, rather than a numeric code. The error message is based on the numeric return code held in the `_IORC_` automatic variable.

This function can only be called from a `DATA` step that has at least one of the following:

- A `MODIFY` function
- A `SET` function with the `KEY` statement

Return type: Character

Example

In this example, it is assumed the catalog specified exists. The result is written to the log.

```

DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  MODIFY example;
  sl = QUOTE(ATTRC(id, 'SORTLVL'));
  PUT sl;
  cl = CLOSE(id);
  msg = IORCMMSG();
  PUT msg;
RUN;
  
```

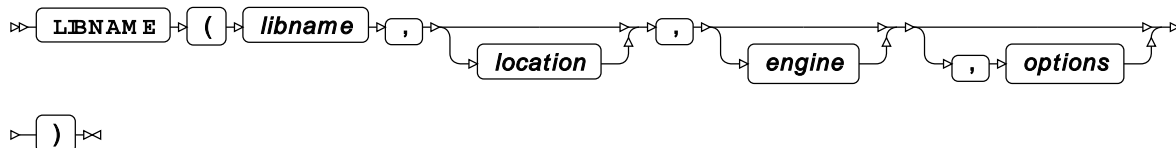
This produces the following output:

```

"STRONG"
The IO operation completed successfully
  
```

LIBNAME

Assigns or deassigns a libref for a library.



Return type: Numeric

Returns 0 (zero) if the library is successfully assigned, otherwise -70008.

libname

Type: Character

The name specified for the library.

location

Optional argument

Type: Character

The location of the library; for example, on Windows this could be a pathname such as C:\data\study1.

engine

Optional argument

Type: Character

An engine type recognised by WPS. For example, WPD, V9SEQ, or SASDASD.

options

Optional argument

Type: Character

An option specific to the dataset. See the LIBNAME statement associated with the engine for more details.

To deassign the libname, specify the function without a library. For example:

```
LIBNAME('ref','C:\Program Files\World Programming WPS 3\sashelp')
```

will assign the libname, while:

```
LIBNAME('ref')
```

will deassign it.

Example – assigning a libname

In this example, a LIBNAME function is used to specify a libname. The result is written to the log.

```
DATA _NULL_;
  lok = LIBNAME('ref','C:\program files\World Programming\WPS\4\sashelp');
  PUT 'Status returned: ' lok;
  id = OPEN('ref.zipcode');
  DO i= 51 TO 55;
    gno = FETCHOBS(id, i);
    nextrec = GETVARN(id,3);
    PUT nextrec;
  END;
  cl = CLOSE(id);
RUN;
```

This produces the following output:

```
Status returned: 0
-66.398866
-66.402557
-66.863322
-66.103857
-66.101442
```

Status returned: 0 indicates that the libname was successfully created for the dataset.

Alternatively, if an attempt was made to create a reference to an unknown library:

```
DATA _NULL_;
  lok = LIBNAME('ref', 'C:\Program Files\World Programming WPS\sashelp');
  PUT 'Status returned: ' lok;
  id = OPEN('ref.zipcode');
  DO i= 51 TO 55;
    gno = FETCHOBS(id, i);
    nextrec = GETVARN(id,3);
    PUT nextrec;
  END;
  cl = CLOSE(id);
RUN;
```

the following status would be returned:

```
Status returned: -70008
```

as well as error message indicating that the arguments to other function are invalid, because no variables had been returned.

Example – deassigning a libname

In this example, a previously assigned libname is deassigned using a LIBNAME function. The result is written to the log.

```
DATA _NULL_;
  lok = LIBNAME('ref');
  PUT 'Status returned: ' lok;
  id = OPEN('ref.zipcode');
  gno = FETCHOBS(id, 55);
  nextrec = GETVARN(id,3);
  PUT nextrec;
  cl = CLOSE(id);
RUN;
```

This produces the following output:

```
Status returned: 0
```

This shows that the `LIBNAME` function executed successfully. However, the following statements and functions return error messages, as no observations and variables are returned; for example:

```
NOTE: Argument 1 to function FETCHOBS at line 1264 column 9 is invalid
NOTE: Argument 1 to function GETVARN at line 1265 column 13 is invalid
.
NOTE: Argument to function CLOSE at line 1267 column 8 is invalid
```

If you were to use the `LIBREF` function to check whether the libname `ref` exists,

```
DATA _NULL_;
  libr = LIBREF('ref');
  PUT 'Has the specified libref been opened: ' libr;
RUN;
```

the following would be returned:

```
Has the specified libref been opened: 70006
```

showing that the libname `ref` no longer has a library name associated with it.

LIBREF

Returns a value indicating whether a specified libname has been assigned.

⇒ **LIBREF** (*argument*) ⇒

Return type: Numeric

If the specified library name has been assigned, 0 (zero) is returned. If the library name does not exist, the error code 70006 is returned.

argument

Type: Character

The libname you want to check.

Example

If the following DATA step had been executed:

```
DATA _NULL_;
  lok = LIBNAME('ref','C:\program files\World Programming\WPS\4\sashelp');
  PUT 'Status returned: ' lok;
  id = OPEN('ref.zipcode');
  DO i= 51 TO 55;
    gno = FETCHOBS(id, i);
    nextrec = GETVARN(id,3);
    PUT nextrec;
  END;
  cl = CLOSE(id);
RUN;
```

then the following example checks whether librefs named `ref` and `reff` have been opened. The result is written to the log.

```
DATA _NULL_;
  libr = LIBREF('ref');
  PUT 'Has the specified libref been opened: ' libr;
  libr = LIBREF('reff');
  PUT 'Has the specified libref been opened: ' libr;
RUN;
```

This produces the following output:

```
Has the specified libref been opened: 0
Has the specified libref been opened: 70006
```

NOTE

Returns an identifier for the current observation from a specified open dataset.

⇒ **NOTE** ⇒ (⇒ *dataset-id* ⇒) ⇒

Return type: Numeric

An identifier for the note. The first note identified will have the identifier 1, the next identified 2, and so on. If the dataset identifier does not exist, 0 (zero) is returned.

dataset-id

Type: Numeric

The identifier of the dataset containing the observation to be noted. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

Example

In this example, the identifier returned by the `OPEN` function is used in the subsequent `NOTE` function. The result is written to the log.

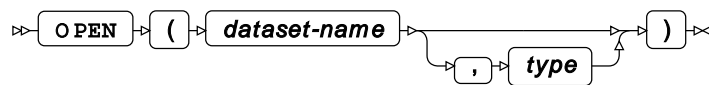
```
DATA _NULL_;  
  lok = LIBNAME('ref','C:\program files\World Programming\WPS\4\sashelp');  
  id = OPEN('ref.zipcode');  
  fo = FETCHOBS(id, 200);  
  DO i= 1 TO 5;  
    gno = FETCH(id);  
    author = GETVARC(id,5);  
  END;  
  obno=NOTE(id);  
  PUT 'The identifier for the observation is: ' obno;  
RUN;
```

This produces the following output:

```
The identifier for the observation is: 1
```

OPEN

Opens a specified dataset, and returns an identifier that can be used by other functions.



Return type: Numeric

If the dataset is not found, 0 is returned. Otherwise a number that will be used as the identifier is returned.

dataset-name

Type: Character

The name of the dataset to be opened.

type

Optional argument

The mode in which the dataset is opened:

"I"

Random access mode.

"IN"

Sequential access mode. Observations are read from beginning to END; previous observations cannot be read.

"IS"

Sequential access mode in which observations are read from beginning to end, but previous observations can be read.

An opened dataset can be closed with the `CLOSE` [↗](#) (page 731) function, specifying the identifier returned by the `OPEN` function. When a dataset is closed, its identifier is released; this identifier will then be used by a subsequent `OPEN`.

Note:

If the `OPEN` returns 0 because the dataset does not exist, using this value in a `CLOSE` function will cause an error.

Example

In this example, the identifiers returned by the `OPEN` function are used in the subsequent `CLOSE` function. The result is written to the log.

```
DATA _NULL_;  
  fo1 = OPEN('sashelp.zipcode', 'IS');  
  fo2 = OPEN('sashelp.mimetypes');  
  PUT 'fo1 identifier: ' fo1;  
  PUT 'fo2 identifier: ' fo2;  
  cr = CLOSE(fo1);  
  cr = CLOSE(fo2);  
RUN;
```

This produces the following output:

```
fo1 identifier: 1  
fo2 identifier: 2
```

POINT

Points at an observation previously identified by the `NOTE` function.

➡ **POINT** (*dataset-id* , *note-id*) ➡

Return type: Numeric

If the observation exists in the dataset, 0 (zero) is returned; 40010 is returned otherwise.

dataset-id

Type: Numeric

The identifier of the dataset containing the noted observation. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

note-id**Type:** Numeric

The identifier generated by the `NOTE` function for the corresponding observation.

Example

In this example, the identifier returned for the dataset by the `OPEN` function, and the identifiers returned for observations by `NOTE` functions, are used to verify whether observations exist. The result is written to the log.

```
DATA _NULL_;
  lok = LIBNAME('ref','C:\program files\World Programming\WPS\4\sashelp');
  id = OPEN('ref.zipcode');

  go = FETCHOBS(id,1);
  vz = GETVARN(id,3);
  PUT vz;
  obno = NOTE(id);

  DO i = 1 TO 100;
    rs= FETCH(id);
  END;
  pt = POINT(id,obno);
  obs = FETCHOBS(id,POINT(id,obno));
  vz = GETVARN(id,3);
  PUT vz;
RUN;
```

This produces the following output:

```
-73.046388
-73.046388
```

The current observation is noted using the `NOTE` function after the first observation is fetched using `FETCHOBS`. A further 100 observations are subsequently read. The observation previously noted is then specified to `POINT`. The next `FETCH` therefore returns the first observation.

REWIND

Positions the dataset pointer at the first record in the dataset.

```
REWIND ( dataset-id )
```

Return type: Numeric***dataset-id*****Type:** Numeric

The identifier of the dataset to rewind. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

Example

In this example, the identifier returned by the `OPEN` function is used in a `REWIND` function. The result is written to the log.

```
DATA _NULL_;  
  id = OPEN('sashelp.zipcode');  
  go = FETCHOBS(id,2010);  
  ob = GETVARN(id,1);  
  PUT 'Observation: ' ob;  
  rc = REWIND(id);  
  go = FETCH(id);  
  ob = GETVARN(id,1);  
  PUT 'Observation: ' ob;  
RUN;
```

This produces the following output:

```
Observation: 5842  
Observation: 501
```

The first result is from the 2010th observation in the dataset; the second result is from the first.

VARFMT

Returns the format of a specified variable, if a format has been applied.

➤ **VARFMT** ➤ (➤ *dataset-id* ➤ , ➤ *variable-index* ➤) ➤ ➤

The function examines the metadata of the dataset for a specified variable to get information about the format. If no format has been applied, a character missing value is returned.

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Type`, in that order, then specifying 3 for this argument would return the format for `Type`.

The value returned will, where applicable, be one of the formats specified in WPS, such as BESTX or F.

Example

In this example, the function is used to return the formats of the variables in a dataset, where those variables have formats applied. The result is written to the log.

```
DATA _NULL_;  
  id = OPEN('sashelp.zipcode');  
  num=ATTRN(id,"nvars");  
  DO i=1 TO num;  
    name = VARNAME(id,i);  
    format = VARFMT(id,i);  
    IF FORMAT = "" THEN PUT name $10. " has no format";  
    ELSE PUT name $10. " has the format " format;  
  END;  
  rc=CLOSE(id);  
RUN;
```

This produces the following output:

```
ZIP          has the format Z5.  
Y            has the format 11.6  
X            has the format 11.6  
ZIP_CLASS    has no format  
CITY         has no format  
STATE        has no format  
STATECODE    has no format  
STATENAME    has no format  
COUNTY      has no format  
COUNTYNM    has no format  
MSA          has no format  
AREACODE     has no format  
TIMEZONE     has no format  
GMTOFFSET    has no format  
DST          has no format
```

In this DATA step, the function is used to return the format for each variable. Only the ZIP, X and Y variables have formats applied.

VARINFMT

Returns the informat of a specified variable, if an informat has been applied.

➤ **VARINFMT** ➤ (➤ *dataset-id* ➤ , ➤ *variable-index* ➤) ➤

The function examines the metadata of the dataset for a specified variable to get information about the informat. If no informat has been applied, a character missing value is returned.

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the `OPEN` [↗](#) (page 752) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Type`, in that order, then specifying 3 for this argument would return the informat for `Type`.

The value returned will, where applicable, be one of the informats available in WPS, such as `INDEXC` or `ANYDTDTE`.

Example

In this example, the function is used to return the formats of the variables in a dataset, where those variables have formats applied. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.vcatalg');
  num=ATTRN(id,"nvars");
  DO i=1 TO num;
    name = VARNAME(id,i);
    format = VARINFMT(id,i);
    IF format="" THEN PUT name $10. " has no informat";
    ELSE PUT name $10. " has the informat " format;
  END;
  rc=CLOSE(id);
RUN;
```

This produces the following output:

libname	has no informat
memname	has no informat
memtype	has no informat
objname	has no informat
objtype	has no informat
objdesc	has no informat
created	has the informat datetime13.
modified	has the informat datetime13.
alias	has no informat

In this `DATA` step, the function is used to return the informat for each variable. Only the `CREATED`, `MODIFIED` and `ALIAS` variables have informats applied.

VARLABEL

Returns the label associated with the specified variable.

➤ **VARLABEL** ➤ (➤ *dataset-id* ➤ , ➤ *variable-index* ➤) ➤

A variable only has a label if one has been specified for it. If there is no label, a character missing value is returned.

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Type`, in that order, then specifying 3 for this argument would return the label for the variable `Type`.

Example

In this example, the function gets the labels of the first and third variables in the specified dataset. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  vl = VARLABEL(id, 1);
  PUT "The Label is: " vl;
  vl = VARLABEL(id, 3);
  PUT "The Label is: " vl;
RUN;
```

This produces the following output:

```
The Label is: The zip code
The Label is: The longitude in degrees of the ZIP code centroid
```

In the following example, the `example` dataset created previously is examined for labels; no labels have been specified, so no information is returned:

```
DATA _NULL_;
  id = OPEN('work.example');
  vn = VARLABEL(id, 1);
  PUT "Variable label is: " vn;
  vn = VARLABEL(id, 2);
  PUT "Variable label is: " vn;
RUN;
```

This produces the following output:

```
Variable label is:  
Variable label is:
```

VARLEN

Returns the length of a specified variable.



Return type: Numeric

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the [OPEN](#) (page 752) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Type`, in that order, then specifying 3 for this argument would return the length of the `Type` variable.

Example

In this example, the function gets the length of the variables that comprise observations in the specified dataset. The result is written to the log.

```
DATA _NULL_;  
  id = OPEN('sashelp.zipcode');  
  num=ATTRN(id,"nvars");  
  DO i=1 TO num;  
    length = VARLEN(id,i);  
    vn = VARNAME(id,i);  
    PUT vn $10. " is " length 3. " characters long" ;  
  END;  
  rc=CLOSE(id);  
RUN;
```

This produces the following output:

```
ZIP          is      8 characters long
Y            is      8 characters long
X            is      8 characters long
ZIP_CLASS    is      1 characters long
CITY         is     64 characters long
STATE        is      8 characters long
STATECODE    is      2 characters long
STATENAME    is     64 characters long
COUNTY      is      8 characters long
COUNTYNM    is     64 characters long
MSA          is      8 characters long
AREACODE     is     16 characters long
TIMEZONE     is     16 characters long
GMTOFFSET    is      8 characters long
DST          is      1 characters long
```

VARNAME

Returns the name of a specified variable. The name is that defined for the variable in the dataset.

➤ **VARNAME** ➤ (➤ *dataset-id* ➤ , ➤ *variable-index* ➤) ➤

Return type: Character

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Type`, in that order, then specifying 3 for this argument would return `Type`.

Example

In this example, the function gets the name of the third variable in the specified dataset. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  vn = VARNAME(id, 5);
  PUT "Variable name is " vn;
RUN;
```

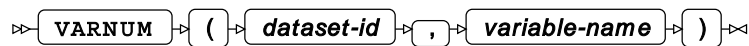
This produces the following output:

```
Variable name is CITY
```

The function returns `city`, which is the name of the fifth variable in an observation in the dataset.

VARNUM

Returns the ordinal position of a named variable in an observation in a dataset.



Return type: Numeric

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the [OPEN](#) (page 752) function.

variable-name

Type: Character

The name of a variable in the dataset.

The *variable-name* must be defined in the dataset; if it is not, 0 will be returned.

Example

In this example, the function gets the position of the specified variable names. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  vn = VARNUM(id, "zip");
  PUT "Variable position is: " vn;
  vn = VARNUM(id, "city");
  PUT "Variable position is: " vn;
  vn = VARNUM(id, "city1");
  PUT "Variable position is: " vn;
RUN;
```

This produces the following output:

```
Variable position is: 1
Variable position is: 5
Variable position is: 0
```

The function returns the ordinal position of the `CITY` and `ZIP` variables in an observation; `CITY1` is not a named variable in the dataset, so the function returns 0.

VARTYPE

Returns the type of the variable at the specified ordinal position in an observation. A variable can be numeric or character.

➤ `VARTYPE` ➤ (➤ *dataset-id* ➤ , ➤ *variable-index* ➤) ➤

Return type: Character

Returns `C` for chracter, `N` for numeric.

dataset-id

Type: Numeric

The identifier of the dataset containing the variable. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

variable-index

Type: Numeric

The ordinal (that is, the index) position of the variable in the observation. The ordinal (that is, the index) position of the variable in the observation. For example, if the observations in the dataset consist of the variables `Author`, `Title` and `Genre`, in that order, then specifying 3 for this argument would return the type of the `Genre` variable.

Example

In this example, the function gets the type of the first and fifth variable in the specified dataset. The result is written to the log.

```
DATA _NULL_;
  id = OPEN('sashelp.zipcode');
  vt = VARTYPE(id, 1);
  PUT "Variable type is: " vt;
  vt = VARTYPE(id, 5);
  PUT "Variable type is: " vt;
RUN;
```

This produces the following output:

```
Variable type is: N
Variable type is: C
```

The function first returns `N`, as the first variable in an observation in the dataset is a zipcode; the second use of the function returns `C`, as the fifth variable of an observation is the name of a city.

CALL SET

Identifies the variables in a dataset, and makes them available to other functions in the `DATA` step. This enables the variable names specified in the dataset to be used in other functions and routines.



dataset-id

Type: Numeric

The identifier of the dataset. This is an identifier generated by the `OPEN` [\(page 752\)](#) function.

Example

In this example, the routine is used to identify the variables in an observation, and these are then used in various `PUT` functions and a string function. The result is written to the log.

```
DATA _NULL_;
  zip=0;
  x=0;
  y=0;
  city=" ";
  state=0;
  statecode=" ";

  id = OPEN('sashelp.zipcode');
  CALL SET(id);

  f = FETCHOBS(id,10);
  PUT zip= x= y=;

  f = FETCHOBS(id,11);
  PUT city= state= statecode=;

  f = FETCHOBS(id,12);
  PUT zip= x= y= city= state= statecode=;

  returnc = CLOSE(id);
RUN;
```

This produces the following output:

```
zip=611 x=-66.797578 y=18.287716
city=A state=72 statecode=P
zip=613 x=-66.719283 y=18.472737 city=A state=72 statecode=P
```

Decision forest functions and CALL routines

Access decision forest functionality.

A *decision forest* takes an ensemble approach to a collection of decision trees to create an approximation to the underlying data.

The prediction produced by a decision forest combines the predictions made by each individual tree. For classification trees the prediction of the forest is the class that gains the most votes from each tree. For a regression tree the prediction of the forest is the mean average of the prediction from each tree. There are multiple ways in which decision forests can be trained, WPS uses the R language randomForest package and the algorithms of Leo Breiman and Adele Cutler (<https://www.stat.berkeley.edu/~breiman/RandomForests/>). The randomForest package must be installed in R before decision forest functionality can be used.

Note:
The term *RandomForest* is a registered trademark of Leo Breiman and Adele Cutler.

Currently only classification trees are supported, and only continuous input variables, that is categorical input variables are not supported. The `exportToWps` within `PROC R` will produce an error if an attempt is made to export a *randomForest* object that is not compatible with the `DF_OPEN` function.

DF_OPEN	764
The function returns an ID that can be used with other decision forest functions and routines.	
DF_PREDICT	765
Produces predictions from a decision forest.	
CALL DF_CLOSE	766
Closes a decision forest previously opened by DF_OPEN.	
CALL DF_DESCRIBE	767
Prints information to the log about the specified decision forest.	
CALL DF_PREDICT	767
Produces predictions from a decision forest, returning multiple result values.	

DF_OPEN

The function returns an ID that can be used with other decision forest functions and routines.

```
>> DF_OPEN ( filename ) <<
```

Opens a decision forest file that was previously generated by the `exportToWps` function within `PROC R`. The `exportToWps` function takes a *randomForest* object produced by the *randomForest* R package (<https://www.stat.berkeley.edu/~breiman/RandomForests/>) and exports the parts necessary for performing prediction. This consists of the definitions of all of the decision trees.

Return type: Numeric

filename

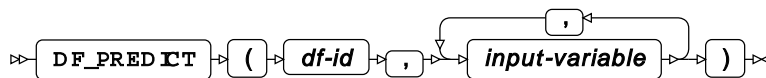
Type: Character

The decision forest file to open.

The function returns an opaque numeric ID. If the forest cannot be opened, then 0 is returned. A specific error code and error text can be retrieved using the `SYSRC` and `SYMSG` functions.

DF_PREDICT

Produces predictions from a decision forest.



For the prediction to be correct, input values must be passed in in the same order as was used during training. The `DF_VARNAME`s call routine can be used to print this list of variables out.

For a classification tree, this returns the predicted class number, where 1 is the first class. The returned class will be the class that received the most votes after evaluating all of the trees in the forest.

For a regression tree, this returns the mean average of the outcome variable.

Return type: Numeric

df-id

Type: Numeric

The ID of a decision forest previously opened using `DF_OPEN`.

input-variable

Type: Numeric

A list of expressions giving the values of the input variables.

Example

```
PROC R;
  SUBMIT;
    data(iris)
      rf = randomForest(iris[,-5], iris[,5], ntree=10)
      exportToWps(rf, "df.dat")
    ENDSUBMIT;
  import r=iris;

PROC FORMAT;
  value species
  1='setosa'
  2='versicolor'
  3='virginica';

DATA predict;
  SET iris;
  RETAIN df;
  IF _n_=1 THEN DO;
    df = DF_OPEN("df.dat");
  END;
  prediction = DF_PREDICT(df,
    Sepal_Length, Sepal_Width, Petal_Length, Petal_Width);
  FORMAT prediction species.;
  KEEP prediction;
RUN;

PROC PRINT data=predict; RUN;
```

CALL DF_CLOSE

Closes a decision forest previously opened by DF_OPEN.

→ **CALL DF_CLOSE** → (*df-id*) → ; →

Note:

It is not necessary to explicitly close decision forests in a DATA step, they are closed automatically when the DATA step ends.

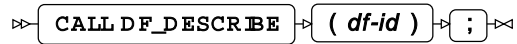
df-id

Type: Numeric

The ID of a decision forest previously opened using DF_OPEN.

CALL DF_DESCRIBE

Prints information to the log about the specified decision forest.



df-id

Type: Numeric

The ID of a decision forest previously opened using DF_OPEN

Example

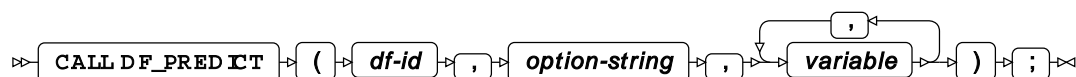
```
PROC R;  
  SUBMIT;  
    DATA(iris)  
    rf = RANDOMFOREST(iris[,-5], iris[,5], ntree=10)  
    EXPORTTOWPS(rf, "df.dat")  
  ENDSUBMIT;  
  
  DATA _NULL_;  
  df = DF_OPEN("df.dat");  
  CALL DF_DESCRIBE(df);  
  RUN;
```

Which produces the following output in the log:

```
NOTE: Decision tree description:  
      Number of trees :          10  
      Number of output classes : 3  
      Input variables :      Sepal.Length, Sepal.Width, Petal.Length,  
      Petal.Width
```

CALL DF_PREDICT

Produces predictions from a decision forest, returning multiple result values.



As with the DF_PREDICT function the order of the input variables is very important to the accuracy of the output.

df-id

Type: Numeric

The ID of a decision forest previously opened using DF_OPEN

option-string

A character string controlling the form of the output. The contents of this string is only read once, so using a string literal is the most appropriate way to pass the value.

For a classification tree, the option string can take the following values. There values are mutually exclusive, and whichever is specified last is used.

"P"

Returns the probabilities for each output class.

"V"

Returns the number of votes cast for each output class.

variable

Type: Numeric

A list of expressions giving the values of the input variables and into which the results of the prediction will be returned.

Example

```
PROC R;
  SUBMIT;
    data(iris)
      rf = randomForest(iris[, -5], iris[, 5], ntree=10)
      exportToWps(rf, "df.dat")
    ENDSUBMIT;
  import r=iris;

  DATA predict;
    SET iris;
    RETAIN df;
    IF _n_=1 THEN DO;
      df = df_open("df.dat");
      CALL DF_DESCRIBE(df);
    END;
    CALL DF_PREDICT(df, 'V', Sepal_Length, Sepal_Width, Petal_Length,
      Petal_Width, votes_setosa, votes_versicolor, votes_virginica);
    KEEP votes;;
  RUN;

  PROC PRINT data=predict; RUN;
```

Difference and lag functions

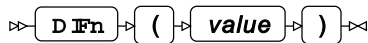
Find the difference or lag between two variables.

DIF ↗	769
Returns the difference between the current value of a specified variable, and the value of that variable in a specified previous observation.	

LAG ↗	772
Returns the value of a specified variable in a specified previous observation.	

DIF

Returns the difference between the current value of a specified variable, and the value of that variable in a specified previous observation.



This function enables you find the difference between the value of a variable in the current observation and the corresponding value in an earlier observation. By default, the difference between the current and previous observation is returned. You can, however, return a value from a previous observation that is a specified number of positions behind the current observation. For example, you could specify that you want to find the difference between the current value of a variable and its value in the previous record (the default), or that you want the difference between the current value and that in an observation that lags six observations behind the current one.

The position of an observation behind the current observation is specified by *n*. This can be set to any integer from 1 to the number of records in the dataset. For example, if you want to get the difference between the value of a variable in the previous record, and the same value in the current record, you would specify `DIF1 (value)`; to get the difference between the value of a variable in an observation that is six positions behind the current observation, and the value in the currently observation, you would specify `DIF6 (value)`. If *n* is omitted, 1 is used; that is, `DIF (value)` is the same as `DIF1 (value)`.

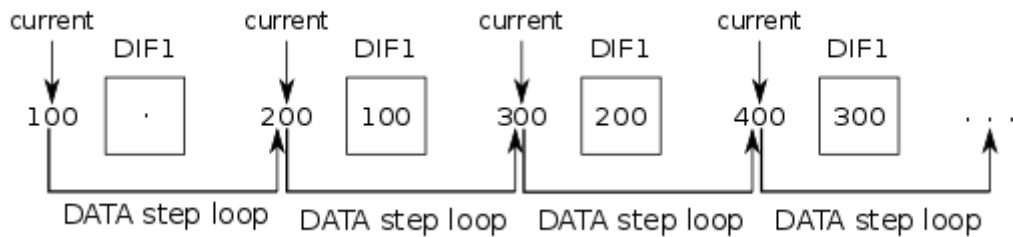
Return type: Character

value

Type: Character or numeric value

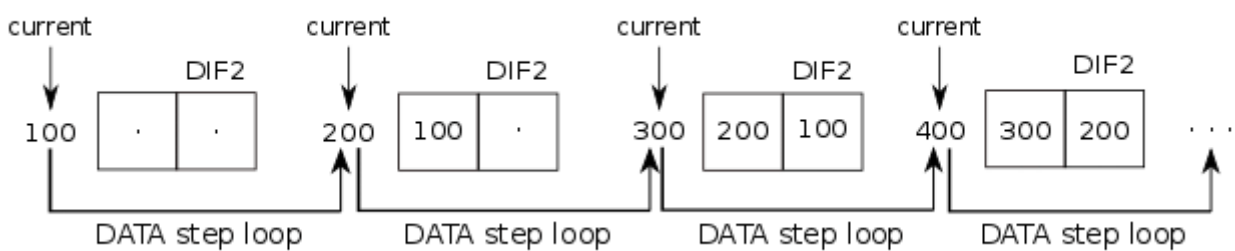
The name of the variable to compare. The variable must contain numeric data. If strings are compared, 0 (zero) is returned.

When a **DATA** step containing a **DIF** function is compiled, a queue is created for each call of the function in the **DATA** step. For example, suppose you specify `DIF1 (value)`, a queue with one position is created. If your dataset contained the values 100, 200, 300, 400, and so on, the queue would be filled as shown in the following diagram:



On the first call to the function, the current value is 100. As there is no preceding observation, the preceding value is missing, and the missing value is returned. On the second call to the function, the queue contains the value 100, from the preceding observation, while the current value is 200. Therefore, the value 100 is returned, which is $200 - 100$.

If you specify `DIF2(value)`, a queue with two positions is created. If you used the same dataset as described above, the queue would be filled as in the following diagram:



Because a queue is created at each call site of the function, the result you obtain might be unexpected. For example, in the following DATA step, the function is invoked by a conditional statement:

```
DATA _NULL_;

  INPUT num1;

  IF num1=500 THEN do;
    dn = DIF(num1);
    PUT dn=;
  END;

datalines;
100
200
300
400
500
;
```

This returns:

```
dn=.
```


The queue for the `DIF` function is only updated at the point the function is invoked, when `num1` equals 500. As this is the first invocation of `DIF` for this call site, there is no previous observation. The queue therefore contains a missing value, and `DIF` returns a missing text value.

Basic example

In this example, the function returns the previous observation. The result is written to the log.

```
DATA _NULL_;

  INPUT num1;

  nf = DIF(num1);

  PUT "The difference between values is: " nf;

datalines;
100
1200
16
23
;
```

This produces the following output:

```
The difference between values is: .
The difference between values is: 1100
The difference between values is: -1184
The difference between values is: 7
```

The first value returned is numeric missing, because there is no value for `num2` before 100.

Because no index is specified on the `DIF` keyword, the value of the variable of the preceding observation is returned (that is, `DIF` is equivalent to `DIF1`).

Example – with different lag

In this example, the function is used to return the difference between the average house price at the end of each year and the house price at the end of the last quarter of each preceding year. The result is written to the log.

```
DATA _NULL_;

  INPUT qrt $ num1;
  pq = DIF4(num1);
  lq = lag4(num1);

  if qrt EQ "Q1-2010" then put "Price at qrt    Price at previous EOY    Difference
between years";

  if substr(qrt, 1, 2) EQ "Q4" then do;
    PUT num1 @16 lq @39 pq;
  end;

datalines;
Q1-2010 162887
```

```

Q2-2010 168719
Q3-2010 167354
Q4-2010 162971
Q1-2011 162379
Q2-2011 166764
Q3-2011 166597
Q4-2011 164785
Q1-2012 162722
Q2-2012 164955
Q3-2012 163910
Q4-2012 162924
Q1-2013 163056
Q2-2013 167294
Q3-2013 170918
Q4-2013 174444
Q1-2014 178124
Q2-2014 186544
Q3-2014 188810
Q4-2014 189002
Q1-2015 188566
Q2-2015 194258
Q3-2015 195733
Q4-2015 197044
Q1-2016 198564
Q2-2016 204238
Q3-2016 206346
Q4-2016 205937
;

```

This produces the following output:

Price at grt	Price at previous EOY	Difference between years
162971	.	.
164785	162971	1814
162924	164785	-1861
174444	162924	11520
189002	174444	14558
197044	189002	8042
205937	197044	8893

The example also uses the `LAG` [↗](#) (page 772) function to display the value at the previous end of year. The first values returned for `DIF` and `LAG` are numeric missing; because the list starts from Q1 2009, there is no value that lags four observations behind Q4 of 2010.

LAG

Returns the value of a specified variable in a specified previous observation.

```

LAGn ( value )

```

You can specify the position of an observation that lags behind the current observation for which you want to return the value of a variable. For example, you could specify that you want the value of a variable in the previous record (the default), or that you want the value in an observation that is six observations behind the current one.

The position of an observation behind the current observation is specified by n . This can be set to any integer from 1 to the number of records in the dataset. For example, if you want to get the value of a variable in the previous record, you would specify `LAG1(value)`; to get the value of a variable in an observation that is six positions behind the current observation, you would specify `LAG6(value)`. If n is omitted, 1 is used; `LAG(value)` is the same as `LAG1(value)`.

Return type: Character

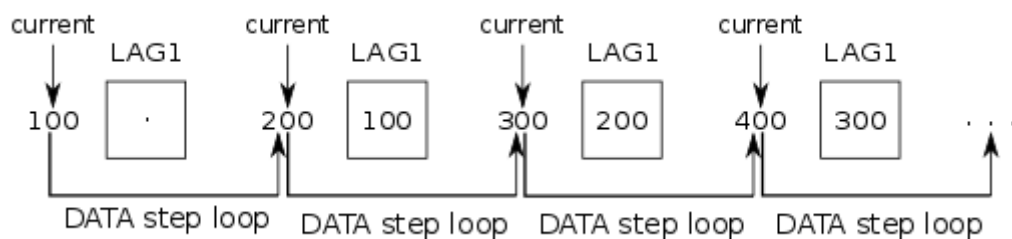
value

Type: Character or numeric value

The name of the variable to be returned.

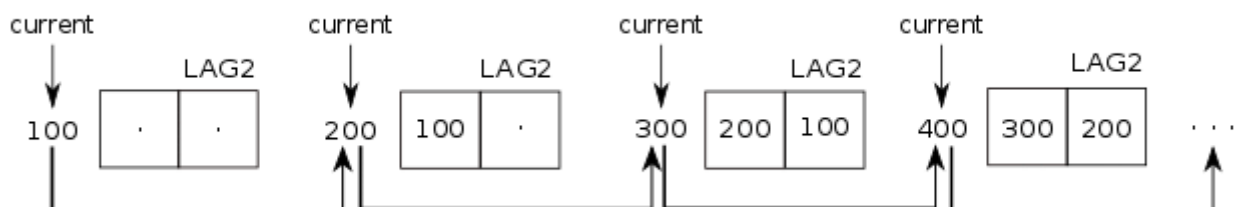
When a DATA step containing a LAG function is compiled, a queue is created for each instance of the function in the DATA step.

For example, suppose you specify `LAG1(value)`, a queue with one position is created. If your dataset contained the values 100, 200, 300, 400, and so on, the queue would be filled as shown in the following diagram:



On the first call to the function, the current value is 100. As there is no preceding observation, the preceding value is missing, and the missing value is returned. On the second call to the function, the queue contains the value 100, from the preceding observation, while the current value is 200. Therefore, the value 100 is returned, as this is the previous value.

If you specify `LAG2(value)`, a queue with two positions is created. If you used the same dataset as described above, the queue would be filled as in the following diagram:



Because a queue is created at each call site of the function, the result you obtain might be unexpected. For example, in the following DATA step, the function is invoked by a conditional statement:

```
LIBNAME books "c:\temp\books";
DATA _NULL_;

    SET books.books=eof;

    IF eof THEN do;
        fa = LAG(author);
        PUT fa=;
    END;

RUN;
```

This returns:

```
fa=
```

The queue for the `LAG` function is only updated at the point it is called. Therefore, the queue only contains the value for `author` that is returned at the end of the file, when the `IF` is triggered. There is no previous observation, so `LAG` returns a missing text value.

Basic example

In this example, the function returns the previous observation. The result is written to the log.

```
DATA _NULL_;

    INPUT num1;

    nf = LAG(num1);

    PUT "The previous value of num1 is " nf;

datalines;
100
1200
16
23
;
```

This produces the following output:

```
The previous value of num1 is .
The previous value of num1 is 100
The previous value of num1 is 1200
The previous value of num1 is 16
```

The first value returned is numeric missing, because there is no value for `num2` before 100.

Because no index is specified on the `LAG` keyword, the value of the variable of the preceding observation is returned (that is, `LAG` is equivalent to `LAG1`).

Example – using LAG to count unique names

In this example, data is read from a dataset of book titles and corresponding author names. The dataset is ordered by author names. The function is used check whether the current value of the `Author` variable is unique by testing it against the value of the variable in the previous observation. If it is unique, a counter is incremented. The result is written to the log.

```
LIBNAME books "c:\temp\books";
DATA _null_;
    retain noa 1;

    SET books.books_author_sorted END=eof;

    IF _N_ > 1 AND author != LAG(author) THEN noa = noa + 1;

    IF eof THEN PUT "Number of unique author names: " noa;

RUN;
```

This produces the following output:

```
Number of unique author names: 1256
```

Distribution-based functions and CALL routines

Perform statistical operations on various probability distributions, including density, survival, quantile and deviance calculations, and drawing random numbers.

CALL STREAMINIT ↗	777
Initialises the random number stream based on a seed.	
Bernoulli distribution ↗	778
Functions for the Bernoulli distribution at a specified point, based on the probability of success.	
Beta distribution ↗	795
Functions for the Beta distribution.	
Binomial distribution ↗	823
Functions and CALL routines for the Binomial distribution at a specified point, based on the probability of success.	
Bivariate Normal distribution ↗	850
Functions for the Bivariate Normal distribution.	
Cauchy distribution ↗	853
Functions and CALL routines for the Cauchy distribution.	
Chi-Squared distribution ↗	874
Functions for the Chi-Squared distribution.	
Erlang distribution ↗	883
Functions for the Erlang distribution.	

Exponential distribution ↗	885
Functions and <code>CALL</code> routines for the Exponential distribution.	
Fisher distribution ↗	905
Functions for the Fisher distribution.	
Gamma distribution ↗	913
Functions and <code>CALL</code> routines for the Gamma distribution.	
Gaussian distribution ↗	943
Functions for the Gaussian distribution.	
Geometric distribution ↗	966
Functions for the Geometric distribution.	
Gumbel distribution ↗	982
Functions for the Gumbel distribution.	
Hypergeometric distribution ↗	1006
Functions for the Hypergeometric distribution.	
Inverse Gaussian distribution ↗	1016
Functions for the Inverse Gaussian distribution.	
Johnson SB distribution ↗	1044
Functions for the Johnson SB distribution.	
Johnson SU distribution ↗	1067
Functions for the Johnson SU distribution.	
Laplace distribution ↗	1091
Functions for the Laplace distribution.	
Logistic distribution ↗	1115
Functions for the Logistic distribution.	
Lognormal distribution ↗	1130
Functions for the Lognormal distribution.	
Negative Binomial distribution ↗	1156
Functions for the Negative Binomial distribution.	
Normal distribution ↗	1176
Functions and <code>CALL</code> routines for the Normal distribution.	
Normal mixture distribution ↗	1209
Functions for the Normal mixture distribution.	
Pareto distribution ↗	1237
Functions for the Pareto distribution.	
Poisson distribution ↗	1245
Functions and <code>CALL</code> routines for the Poisson distribution.	
Power distribution ↗	1274
Functions for the Power distribution.	
Rayleigh distribution ↗	1282
Functions for the Rayleigh distribution.	

Student's T distribution ↗	1283
Functions for the Student's T distribution.	
Table distribution ↗	1291
Functions and <code>CALL</code> routines for the Table distribution.	
Triangular distribution ↗	1302
Functions and <code>CALL</code> routines for the Triangular distribution.	
Tweedie distribution ↗	1308
Functions for the Tweedie distribution.	
Uniform distribution ↗	1314
Functions and <code>CALL</code> routines for the Uniform distribution.	
Wald distribution ↗	1335
Functions for the Wald distribution.	
Weibull distribution ↗	1360
Functions for the Weibull distribution.	

CALL STREAMINIT

Initialises the random number stream based on a seed.



To initialise the stream of random numbers, this routine must be executed before any other random number function or routine within the same `DATA` step. Only the first call to this routine within a `DATA` step initialises the random stream; all subsequent calls are ignored.

To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, the function is used to generate a seed for `RAND`.

```
DATA _NULL_;
  CALL STREAMINIT(100);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND ("BINOMIAL", 0.2, 50);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
14
12
7
10
7
```

Running the `DATA` step again produces the same output.

Bernoulli distribution

Functions for the Bernoulli distribution at a specified point, based on the probability of success.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – `BERNOULLI` [↗](#)..... 780

$$\begin{cases} \text{if } x=1 & \text{return } p \\ \text{if } x=0 & \text{return } 1-p \\ \text{otherwise} & \text{return } 0 \end{cases}$$

Returns the probability density of the Bernoulli distribution, based on the probability of success. This function is an alias of `PMF – BERNOLLI`.

PMF – `BERNOULLI` [↗](#)..... 781

$$\begin{cases} \text{if } x=1 & \text{return } p \\ \text{if } x=0 & \text{return } 1-p \\ \text{otherwise} & \text{return } 0 \end{cases}$$

Returns the probability mass of the Bernoulli distribution, based on the probability of success. This function is an alias of `PDF – BERNOLLI`.

LOGPDF – BERNOULLI ↗	782
$\begin{cases} \text{if } x=0 \text{ and } p < 1 & \text{return } \log(1-p) \\ \text{if } x=1 \text{ and } p > 0 & \text{return } \log(p) \end{cases}$	
Returns the natural logarithm of the probability density of the Bernoulli distribution, based on the probability of success. This function is an alias of LOGPMF – BERNOULLI.	
LOGPMF – BERNOULLI ↗	783
$\begin{cases} \text{if } x=0 \text{ and } p < 1 & \text{return } \log(1-p) \\ \text{if } x=1 \text{ and } p > 0 & \text{return } \log(p) \end{cases}$	
Returns the natural logarithm of the probability mass of the Bernoulli distribution, based on the probability of success. This function is an alias of LOGPDF – BERNOULLI.	
CDF – BERNOULLI ↗	784
$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{if } 0 \leq x < 1 & \text{return } 1-p \\ \text{if } x \geq 1 & \text{return } 1 \end{cases}$	
Returns the cumulative density of the Bernoulli distribution, based on the probability of success.	
LOGCDF – BERNOULLI ↗	786
$\begin{cases} \text{if } 0 \leq x < 1 & \text{return } \log(1-p) \\ \text{if } x \geq 1 & \text{return } 0 \end{cases}$	
Returns the natural logarithm of the cumulative density of the Bernoulli distribution, based on the probability of success.	
SDF – BERNOULLI ↗	787
$\begin{cases} \text{if } x < 0 & \text{return } 1 \\ \text{if } x \geq 1 & \text{return } 0 \\ \text{otherwise} & \text{return } p \end{cases}$	
Returns the survival of the Bernoulli distribution, based on the probability of success.	
LOGSDF – BERNOULLI ↗	788
$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{if } x \geq 0 \text{ and } p > 0 & \text{return } \log(p) \end{cases}$	
Returns the natural logarithm of the survival of the Bernoulli distribution, based on the probability of success.	
QUANTILE – BERNOULLI ↗	789
$\begin{cases} \text{if } q \leq 1-p & \text{return } 0 \\ \text{otherwise} & \text{return } 1 \end{cases}$	
Returns the quantile of the Bernoulli distribution for a specified probability value, based on the probability of success.	
DEVIANCE – BERNOULLI ↗	790
$\begin{cases} \text{if } x=0 & \text{return } -2\log(1-p) \\ \text{if } x=1 & \text{return } -2\log(p) \end{cases}$	
Returns the deviance of the Bernoulli distribution at a specified point, based on the probability of success.	
RAND – BERNOULLI ↗	793
Returns a random number from the Bernoulli distribution based on the probability of success.	

PDF – BERNOULLI

Returns the probability density of the Bernoulli distribution, based on the probability of success. This function is an alias of PMF – BERNOULLI.

➤ PDF ➤ (➤ "BERNOULLI" ➤ , ➤ x ➤ , ➤ p ➤) ➤ ➤

Calculates the probability density function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$0 \leq p \leq 1$$

$$\begin{cases} \text{if } x=1 & \text{return } p \\ \text{if } x=0 & \text{return } 1-p \\ \text{otherwise} & \text{return } 0 \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF("BERNOULLI", 0, 0.7);
  PUT s1=;
  s2 = PDF("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = PDF("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = PDF("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=0.3
s2=0
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

PMF – BERNOULLI

Returns the probability mass of the Bernoulli distribution, based on the probability of success. This function is an alias of PDF – BERNOULLI.

```
PMF ( "BERNOULLI" , x , p )
```

Calculates the probability mass function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$0 \leq p \leq 1$$

$$\begin{cases} \text{if } x=1 & \text{return } p \\ \text{if } x=0 & \text{return } 1-p \\ \text{otherwise} & \text{return } 0 \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("BERNOULLI", 0, 0.7);
  PUT s1=;
  s2 = PMF ("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = PMF ("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = PMF ("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=0.3
s2=0
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGPDF – BERNOLLI

Returns the natural logarithm of the probability density of the Bernoulli distribution, based on the probability of success. This function is an alias of LOGPMF – BERNOLLI.

```
LOGPDF ( "BERNOULLI" , x , p )
```

Calculates the natural logarithm of the probability density function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$\begin{aligned} & x = 0 \text{ or } x = 1 \\ & 0 \leq p < 1 \\ & \begin{cases} \text{if } x = 0 \text{ and } p < 1 & \text{return } \log(1-p) \\ \text{if } x = 1 \text{ and } p > 0 & \text{return } \log(p) \end{cases} \end{aligned}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

Restriction: $x = 0$ or $x = 1$

If the argument is out of range, a missing value is returned.

p **Type:** Numeric

The probability of success for each trial.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("BERNOULLI", 0, 0.7);
  PUT s1=;
  s2 = LOGPDF ("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = LOGPDF ("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = LOGPDF ("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=-1.203972804
s2=.
s3=.
s4=.
```

The last three examples return a missing value because one of the arguments is out of range.

LOGPMF – BERNOULLI

Returns the natural logarithm of the probability mass of the Bernoulli distribution, based on the probability of success. This function is an alias of LOGPDF – BERNOULLI.

```
LOGPMF ( "BERNOULLI" , x , p )
```

Calculates the natural logarithm of the probability mass function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$\begin{aligned} & x = 0 \text{ or } x = 1 \\ & 0 \leq p < 1 \\ & \begin{cases} \text{if } x = 0 \text{ and } p < 1 & \text{return } \log(1-p) \\ \text{if } x = 1 \text{ and } p > 0 & \text{return } \log(p) \end{cases} \end{aligned}$$

Return type: Numeric

x **Type:** Numeric

The point at which to calculate the natural logarithm of the probability mass.

Restriction: $x = 0$ or $x = 1$

If the argument is out of range, a missing value is returned.

 p **Type:** Numeric

The probability of success for each trial.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPMF ("BERNOULLI", 0, 0.7);  
  PUT s1=;  
  s2 = LOGPMF ("BERNOULLI", -1, 0.7);  
  PUT s2=;  
  s3 = LOGPMF ("BERNOULLI", 0, 1.7);  
  PUT s3=;  
  s4 = LOGPMF ("BERNOULLI", 0, -1.7);  
  PUT s4=;  
RUN;
```

This produces the following output:

```
s1=-1.203972804  
s2=.  
s3=.  
s4=.
```

The last three examples return a missing value because one of the arguments is out of range.

CDF – BERNOULLI

Returns the cumulative density of the Bernoulli distribution, based on the probability of success.

➤ CDF ➤ (➤ "BERNOULLI" ➤ , ➤ x ➤ , ➤ p ➤) ➤

Calculates the cumulative density function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$0 \leq p \leq 1$$
$$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{if } 0 \leq x < 1 & \text{return } 1 - p \\ \text{if } x \geq 1 & \text{return } 1 \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF("BERNOULLI", 0, 0.7);
  PUT s1=;
  s2 = CDF("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = CDF("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = CDF("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

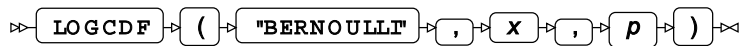
This produces the following output:

```
s1=0.3
s2=0
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGCDF – BERNOULLI

Returns the natural logarithm of the cumulative density of the Bernoulli distribution, based on the probability of success.



Calculates the natural logarithm of the cumulative density function for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$\begin{aligned}
 &x \geq 0, \quad 0 \leq p < 1 \\
 &\begin{cases} \text{if } 0 \leq x < 1 & \text{return } \log(1-p) \\ \text{if } x \geq 1 & \text{return } 0 \end{cases}
 \end{aligned}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF("BERNOULLI", 0, 0.7);
  PUT s1=;
  s2 = LOGCDF("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = LOGCDF("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = LOGCDF("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```


This produces the following output:

```
s1=-1.203972804
s2=.
s3=.
s4=.
```

The last three examples return a missing value because one of the arguments is out of range.

SDF – BERNOULLI

Returns the survival of the Bernoulli distribution, based on the probability of success.



Calculates the survival, or the complement to the cumulative density function, for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$0 \leq p \leq 1$$

$$\begin{cases} \text{if } x < 0 & \text{return } 1 \\ \text{if } x \geq 1 & \text{return } 0 \\ \text{otherwise} & \text{return } p \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("BERNOULLI", 1, 0.7);
  PUT s1=;
  s2 = SDF ("BERNOULLI", -1, 0.7);
  PUT s2=;
  s3 = SDF ("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = SDF ("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=0
s2=1
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGSDF – BERNOULLI

Returns the natural logarithm of the survival of the Bernoulli distribution, based on the probability of success.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Bernoulli distribution at point x , based on the probability of success p .

This function is defined under the following conditions:

$$x < 1, 0 \leq p < 1$$

$$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{if } x \geq 0 \text{ and } p > 0 & \text{return } \log(p) \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

Restriction: $x < 1$

If the argument is out of range, a missing value is returned.

p **Type:** Numeric

The probability of success for each trial.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF ("BERNOULLI", -1, 0.7);
  PUT s1=;
  s2 = LOGSDF ("BERNOULLI", 1, 0.7);
  PUT s2=;
  s3 = LOGSDF ("BERNOULLI", 0, 1.7);
  PUT s3=;
  s4 = LOGSDF ("BERNOULLI", 0, -1.7);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=0
s2=.
s3=.
s4=.
```

The last three examples return a missing value because one of the arguments is out of range.

QUANTILE – BERNOULLI

Returns the quantile of the Bernoulli distribution for a specified probability value, based on the probability of success.

```
QUANTILE ( "BERNOULLI" , q , p )
```

Calculates the quantile x , or the inverse of the cumulative density function, for the Bernoulli distribution for probability value q based on the probability of success p .

The quantile function returns point x such that randomly drawn values from the distribution fall below x with probability q .

This function is defined under the following conditions:

$$0 < q < 1, 0 \leq p \leq 1$$

$$\begin{cases} \text{if } q \leq 1-p & \text{return } 0 \\ \text{otherwise} & \text{return } 1 \end{cases}$$

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Bernoulli distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = QUANTILE("BERNOULLI", 0.7, 0);  
  PUT s1=;  
  s2 = QUANTILE("BERNOULLI", -1, 0.7);  
  PUT s2=;  
  s3 = QUANTILE("BERNOULLI", 0, 1.7);  
  PUT s3=;  
  s4 = QUANTILE("BERNOULLI", 0, -1.7);  
  PUT s4=;  
RUN;
```

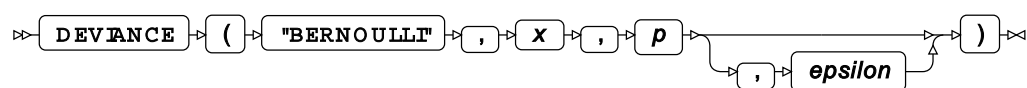
This produces the following output:

```
s1=0  
s2=.  
s3=.  
s4=.
```

The last three examples return a missing value because one of the arguments is out of range.

DEVIANCE – BERNOLLI

Returns the deviance of the Bernoulli distribution at a specified point, based on the probability of success.



Calculates the deviance, or goodness of fit, for the generalised linear model of the Bernoulli distribution at point x based on the probability of success p . An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The probability of success is then adjusted so that $\varepsilon \leq p \leq 1-\varepsilon$:

$$\begin{cases} \text{if } p < \varepsilon & \text{set } p = \varepsilon \\ \text{if } p > 1-\varepsilon & \text{set } p = 1-\varepsilon \end{cases}$$

This adjusted value of p is used in the subsequent calculation of the deviance.

$$\begin{cases} \text{if } x=0 & \text{return } -2\log(1-p) \\ \text{if } x=1 & \text{return } -2\log(p) \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $x=0$ or $x=1$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Expected: $0 < p < 1$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$

Expected: $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Examples – applying correction to the probability of success

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("BERNOULLI", 1, 0.99992, 0.00005);  
  PUT g1=;  
  g2 = DEVIANCE("BERNOULLI", 1, 0.99992, 0.00010);  
  PUT g2=;  
  g3 = DEVIANCE("BERNOULLI", 1, 0.99992, 0.00015);  
  PUT g3=;  
  g4 = DEVIANCE("BERNOULLI", 1, 0.99992          );  
  PUT g4=;  
RUN;
```

This produces the following output:

```
p1=0.99992  
p2=0.99990  
p3=0.99985  
p4=0.99992  
  
g1=0.0001600064  
g2=0.0002000100  
g3=0.0003000225  
g4=0.0001600064
```

The value of the probability of success is not corrected in the first example because $p < 1 - \varepsilon$. However, this condition does not hold in the second and third example, and correction is applied: $p = 1 - \varepsilon$. This corrected value is used for calculation, yielding different results.

In the fourth example the ε parameter is omitted, so the default value of $\varepsilon = 10^{-12}$ is used. Here, as in the first example, $p < 1 - \varepsilon$, so no correction is required.

Examples – deviance calculation

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("BERNOULLI", 0, 0.7);  
  PUT g1=;  
  g2 = DEVIANCE("BERNOULLI", 0, 0.5);  
  PUT g2=;  
  g3 = DEVIANCE("BERNOULLI", 0, 0.3);  
  PUT g3=;  
  g4 = DEVIANCE("BERNOULLI", 1, 0.7);  
  PUT g4=;  
  g5 = DEVIANCE("BERNOULLI", 1, 0.5);  
  PUT g5=;  
  g6 = DEVIANCE("BERNOULLI", 1, 0.3);  
  PUT g6=;  
RUN;
```

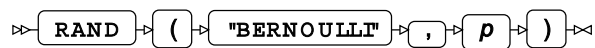
This produces the following output:

```
g1=2.4079456087
g2=1.3862943611
g3=0.7133498879
g4=0.7133498879
g5=1.3862943611
g6=2.4079456087
```

For the Bernoulli distribution, the deviance is symmetrical around $p=0.5$ for the opposite values of x . Thus, in the above example $g1 = g6$, $g2 = g5$, and $g3 = g4$.

RAND – BERNOULLI

Returns a random number from the Bernoulli distribution based on the probability of success.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

The return value is either 0 or 1.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Example

In this example, a random number from the Bernoulli distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("BERNOULLI", 0.75);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0  
0  
1  
0  
1
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0  
1  
0  
0  
1
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(10);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("BERNOULLI", 0.75);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
1  
0  
0  
1  
1
```

Running the DATA step again produces the same output.

Beta distribution

Functions for the Beta distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – BETA [↗](#)..... 796

$$\frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}$$

Returns the probability density of the Beta distribution, based on the shape parameters α and β . This function is an alias of PMF – BETA.

PMF – BETA [↗](#)..... 799

$$\frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}$$

Returns the probability mass of the Beta distribution, based on the shape parameters α and β . This function is an alias of PDF – BETA.

LOGPDF – BETA [↗](#)..... 801

$$\log \frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}$$

Returns the natural logarithm of the probability density of the Beta distribution, based on the shape parameters α and β . This function is an alias of LOGPMF – BETA.

LOGPMF – BETA [↗](#)..... 803

$$\log \frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}$$

Returns the natural logarithm of the probability mass of the Beta distribution, based on the shape parameters α and β . This function is an alias of LOGPDF – BETA.

CDF – BETA [↗](#)..... 806

$$I_z(\alpha, \beta) = \frac{B(z; \alpha, \beta)}{B(\alpha, \beta)}$$

Returns the cumulative density of the Beta distribution, based on the shape parameters α and β . This function is similar to PROBBETA where optional arguments are set to default values.

PROBBETA [↗](#)..... 808

$$I_x(\alpha, \beta) = \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)}$$

Returns the cumulative density of the Beta distribution, based on the shape parameters α and β .
This function is similar to CDF – BETA where optional arguments are set to default values.

LOGCDF – BETA [↗](#)..... 810

$$\log I_z(\alpha, \beta) = \log B(z; \alpha, \beta) - \log B(\alpha, \beta)$$

Returns the natural logarithm of the cumulative density of the Beta distribution, based on the shape parameters α and β .

SDF – BETA [↗](#)..... 812

$$1 - I_z(\alpha, \beta)$$

Returns the survival of the Beta distribution, based on the shape parameters α and β .

LOGSDF – BETA [↗](#)..... 815

$$\log[1 - I_z(\alpha, \beta)]$$

Returns the natural logarithm of the survival of the Beta distribution, based on the shape parameters α and β .

QUANTILE – BETA [↗](#)..... 817

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

Returns the quantile of the Beta distribution for a specified probability value, based on the shape parameters α and β . This function is similar to BETAINV where optional arguments are set to default values.

BETAINV [↗](#)..... 819

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

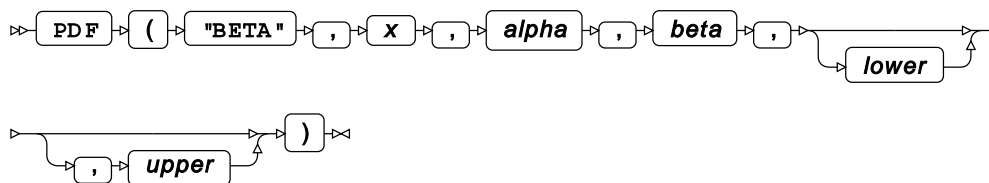
Returns the quantile of the Beta distribution for a specified probability value, based on the shape parameters α and β . This function is similar to QUANTILE – BETA where optional arguments are set to default values.

RAND – BETA [↗](#)..... 821

Returns a random number from the Beta distribution based on the shape parameters α and β .

PDF – BETA

Returns the probability density of the Beta distribution, based on the shape parameters α and β . This function is an alias of PMF – BETA.



Calculates the probability density function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*). The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

This function is defined under the following conditions:

$$\begin{aligned}
 & \text{lower} < \text{upper} \\
 & \begin{cases} \text{if } x = \text{lower} \text{ then } \alpha > 1 \\ \text{if } x = \text{upper} \text{ then } \beta > 1 \\ \text{otherwise} \quad \quad \quad \alpha > 0, \beta > 0 \end{cases} \\
 & \begin{cases} \text{if } x < \text{lower} \text{ or } x > \text{upper} \text{ return } 0 \\ \text{if } x = \text{lower} \text{ and } \alpha > 1 \text{ return } 0 \\ \text{if } x = \text{upper} \text{ and } \beta > 1 \text{ return } 0 \\ \text{if } \text{lower} \leq x \leq \text{upper} \text{ return } f(x; \alpha, \beta) \end{cases} \\
 & f(x; \alpha, \beta) = \frac{(\text{x} - \text{lower})^{\alpha-1} (\text{upper} - \text{x})^{\beta-1}}{\text{B}(\alpha, \beta)} \\
 & \text{B}(\alpha, \beta) = \int_{\text{lower}}^{\text{upper}} t^{\alpha-1} (1-t)^{\beta-1} dt
 \end{aligned}$$

where $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction:

if $x = \text{lower}$, then $\alpha > 1$
otherwise $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction:

if $x = \text{upper}$, then $\beta > 1$
otherwise $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = PDF ("BETA", 0.7, 0.7, 3, -1, 5);  
  PUT s1=;  
  s2 = PDF ("BETA", 2.7, 0.7, 3, 1, 7);  
  PUT s2=;  
  s3 = PDF ("BETA", -1, 1.7, 3, -1, 5);  
  PUT s3=;  
  s4 = PDF ("BETA", -1, 0.7, 3, -1, 5);  
  PUT s4=;  
RUN;
```

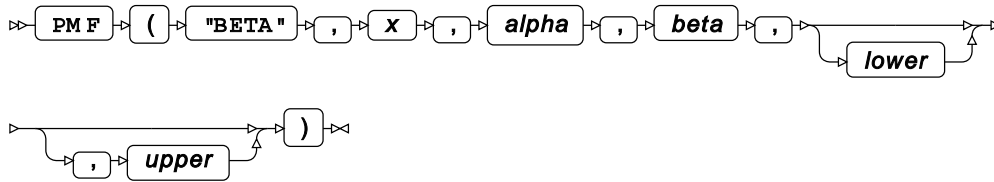
This produces the following output:

```
s1=0.2007587988  
s2=0.2007587988  
s3=0  
s4=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , $lower$ and $upper$ are increased or decreased by the same value. The last example returns a missing value because one of the arguments is out of range.

PMF – BETA

Returns the probability mass of the Beta distribution, based on the shape parameters α and β . This function is an alias of PDF – BETA.



Calculates the probability mass function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*). The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

This function is defined under the following conditions:

$$\begin{aligned}
 & \text{lower} < \text{upper} \\
 & \begin{cases} \text{if } x = \text{lower} \text{ then } \alpha > 1 \\ \text{if } x = \text{upper} \text{ then } \beta > 1 \\ \text{otherwise} \quad \alpha > 0, \beta > 0 \end{cases} \\
 & \begin{cases} \text{if } x < \text{lower} \text{ or } x > \text{upper} & \text{return } 0 \\ \text{if } x = \text{lower} \text{ and } \alpha > 1 & \text{return } 0 \\ \text{if } x = \text{upper} \text{ and } \beta > 1 & \text{return } 0 \\ \text{if } \text{lower} \leq x \leq \text{upper} & \text{return } f(x; \alpha, \beta) \end{cases} \\
 & f(x; \alpha, \beta) = \frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}
 \end{aligned}$$

$$B(\alpha, \beta) = \int_{\text{lower}}^{\text{upper}} t^{\alpha-1} (1-t)^{\beta-1} dt$$

where $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction:

if $x = \text{lower}$, then $\alpha > 1$

otherwise $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction:

if $x = \text{upper}$, then $\beta > 1$

otherwise $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If lower is specified, upper must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $\text{upper} > \text{lower}$

If upper is specified, lower must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("BETA", 0.7, 0.7, 3, -1, 5);
  PUT s1=;
  s2 = PMF ("BETA", 2.7, 0.7, 3, 1, 7);
  PUT s2=;
  s3 = PMF ("BETA", -1, 1.7, 3, -1, 5);
  PUT s3=;
  s4 = PMF ("BETA", -1, 0.7, 3, -1, 5);
  PUT s4=;
RUN;
```

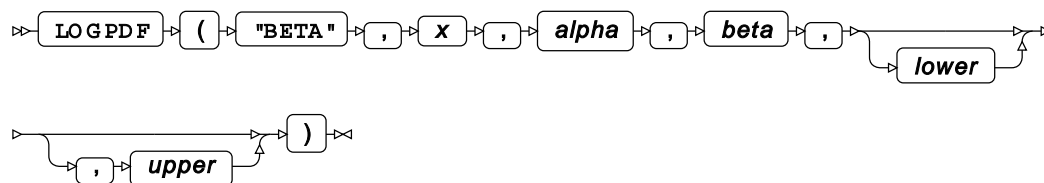
This produces the following output:

```
s1=0.2007587988
s2=0.2007587988
s3=0
s4=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last example returns a missing value because one of the arguments is out of range.

LOGPDF – BETA

Returns the natural logarithm of the probability density of the Beta distribution, based on the shape parameters α and β . This function is an alias of LOGPMF – BETA.



Calculates the natural logarithm of the probability density function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*). The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

This function is defined under the following conditions:

$$\alpha > 0, \beta > 0, lower < x < upper$$

$$f(x; \alpha, \beta) = \log \frac{(x - lower)^{\alpha-1} (upper - x)^{\beta-1}}{B(\alpha, \beta)}$$

$$= (\alpha - 1) \log(x - lower) + (\beta - 1) \log(upper - x) - \log B(\alpha, \beta)$$

$$B(\alpha, \beta) = \int_{lower}^{upper} t^{\alpha-1} (1-t)^{\beta-1} dt$$

where $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

Restriction: $lower < x < upper$

If the argument is out of range, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("BETA", 0.7, 0.7, 3, -1, 5);
  PUT s1=;
  s2 = LOGPDF ("BETA", 2.7, 0.7, 3, -1, 5);
  PUT s2=;
  s3 = LOGPDF ("BETA", -1, 1.7, 3, -1, 5);
  PUT s3=;
RUN;
```

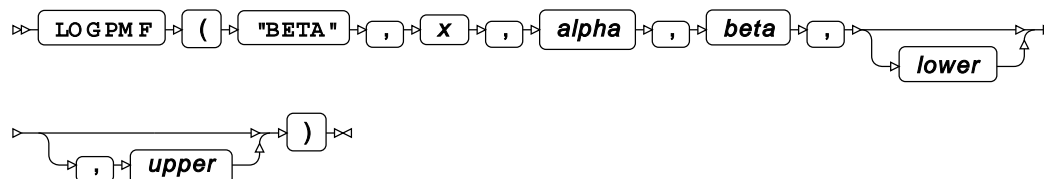
This produces the following output:

```
s1=-1.605651097
s2=-1.605651097
s3=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , $lower$ and $upper$ are increased or decreased by the same value. The last two examples return a missing value because one of the arguments is out of range.

LOGPMF – BETA

Returns the natural logarithm of the probability mass of the Beta distribution, based on the shape parameters α and β . This function is an alias of LOGPDF – BETA.



Calculates the natural logarithm of the probability mass function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*). The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

This function is defined under the following conditions:

$$\alpha > 0, \beta > 0, \text{lower} < x < \text{upper}$$

$$f(x; \alpha, \beta) = \log \frac{(x - \text{lower})^{\alpha-1} (\text{upper} - x)^{\beta-1}}{B(\alpha, \beta)}$$

$$= (\alpha - 1) \log(x - \text{lower}) + (\beta - 1) \log(\text{upper} - x) - \log B(\alpha, \beta)$$

$$B(\alpha, \beta) = \int_{\text{lower}}^{\text{upper}} t^{\alpha-1} (1-t)^{\beta-1} dt$$

where $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

Restriction: $\text{lower} < x < \text{upper}$

If the argument is out of range, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPMF ("BETA", 0.7, 0.7, 3, -1, 5);  
  PUT s1=;  
  s2 = LOGPMF ("BETA", 2.7, 0.7, 3, -1, 5);  
  PUT s2=;  
  s3 = LOGPMF ("BETA", -1, 1.7, 3, -1, 5);  
  PUT s3=;  
RUN;
```

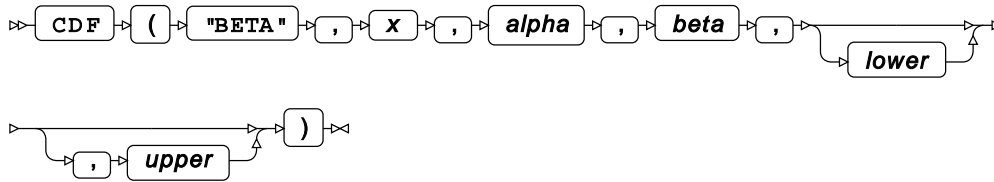
This produces the following output:

```
s1=-1.605651097  
s2=-1.605651097  
s3=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last two examples return a missing value because one of the arguments is out of range.

CDF – BETA

Returns the cumulative density of the Beta distribution, based on the shape parameters α and β . This function is similar to PROBBETA where optional arguments are set to default values.



Calculates the cumulative density function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*).

The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted. Prior to computation, the value of x is normalised with respect to the domain bounds as follows:

$$z = \frac{x - \text{lower}}{\text{upper} - \text{lower}}$$

This normalised value is then used in further calculations.

This function is defined under the following conditions:

$$\text{lower} < \text{upper}$$

$$\begin{cases} \text{if } z \leq 0 & \text{return } 0 \\ \text{if } z \geq 1 & \text{return } 1 \\ \text{otherwise} & \text{return } I_z(\alpha, \beta) \end{cases}$$

$$I_z(\alpha, \beta) = \frac{B(z; \alpha, \beta)}{B(\alpha, \beta)}$$

$$B(z; \alpha, \beta) = \int_0^z t^{\alpha-1} (1-t)^{\beta-1} dt$$

$$B(\alpha, \beta) = B(z; \alpha, \beta) \Big|_{z=1}$$

where $I_z(\alpha, \beta)$ is the regularised incomplete Beta function; $B(z; \alpha, \beta)$ is the incomplete Beta function; and $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $\text{upper} > \text{lower}$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("BETA", 0.7, 0.7, 3, -1, 5);
  PUT s1=;
  s2 = CDF ("BETA", 2.7, 0.7, 3, 1, 7);
  PUT s2=;
  s3 = CDF ("BETA", 7, 0.7, 3, -1, 5);
  PUT s3=;
  s4 = CDF ("BETA", -3, 0.7, 3, -1, 5);
  PUT s4=;
  s5 = CDF ("BETA", 0, -0.7, 3, -1, 5);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.7475328365
s2=0.7475328365
s3=1
s4=0
s5=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last example returns a missing value because one of the arguments is out of range.

PROBBETA

Returns the cumulative density of the Beta distribution, based on the shape parameters α and β . This function is similar to CDF – BETA where optional arguments are set to default values.

```
PROBBETA ( x , alpha , beta )
```

Calculates the cumulative density function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*).

This function is defined under the following conditions:

$$0 < x < 1, \alpha > 0, \beta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x \geq 1 & \text{return } 1 \\ \text{otherwise} & \text{return } I_x(\alpha, \beta) \end{cases}$$

$$I_x(\alpha, \beta) = \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)}$$

$$B(x; \alpha, \beta) = \int_0^x t^{\alpha-1} (1-t)^{\beta-1} dt$$

$$B(\alpha, \beta) = B(x; \alpha, \beta)|_{x=1}$$

where $I_x(\alpha, \beta)$ is the regularised incomplete Beta function; $B(x; \alpha, \beta)$ is the incomplete Beta function; and $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

Restriction: $0 < x < 1$

If the argument is out of range, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $beta > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PROBBETA (0,0.7,3);
  PUT s1=;
  s2 = PROBBETA (1,0.7,3);
  PUT s2=;
  s3 = PROBBETA (2,0.7,3);
  PUT s3=;
RUN;
```

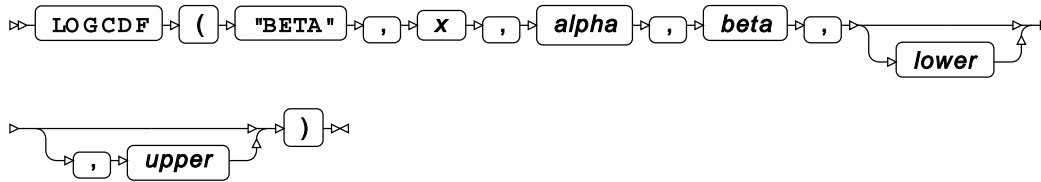
This produces the following output:

```
s1=0
s2=1
s3=.
```

The last example returns a missing value because one of the arguments is out of range.

LOGCDF – BETA

Returns the natural logarithm of the cumulative density of the Beta distribution, based on the shape parameters α and β .



Calculates the natural logarithm of the cumulative density function for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*).

The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted. Prior to computation, the value of x is normalised with respect to the domain bounds as follows:

$$z = \frac{x - \text{lower}}{\text{upper} - \text{lower}}$$

This normalised value is then used in further calculations.

This function is defined under the following conditions:

$$\text{lower} < \text{upper}, x > \text{lower}$$

$$\begin{cases} \text{if } z \geq 1 & \text{return } 0 \\ \text{if } 0 < z < 1 & \text{return } f(z; \alpha, \beta) \end{cases}$$

$$f(z; \alpha, \beta) = \log I_z(\alpha, \beta) = \log B(z; \alpha, \beta) - \log B(\alpha, \beta)$$

$$B(z; \alpha, \beta) = \int_0^z t^{\alpha-1} (1-t)^{\beta-1} dt$$

$$B(\alpha, \beta) = B(z; \alpha, \beta) \Big|_{z=1}$$

where $I_z(\alpha, \beta)$ is the regularised incomplete Beta function; $B(z; \alpha, \beta)$ is the incomplete Beta function; and $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: $x > \text{lower}$

If the argument is out of range, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $\text{upper} > \text{lower}$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("BETA", 0.7, 0.7, 3, -1, 5);
  PUT s1=;
  s3 = LOGCDF ("BETA", 2.7, 0.7, 3, 1, 7);
  PUT s3=;
  s4 = LOGCDF ("BETA", 7, 0.7, 3, -1, 5);
  PUT s4=;
  s5 = LOGCDF ("BETA", -3, 0.7, 3, -1, 5);
  PUT s5=;
  s6 = LOGCDF ("BETA", 0, -0.7, 3, -1, 5);
  PUT s6=;
RUN;
```

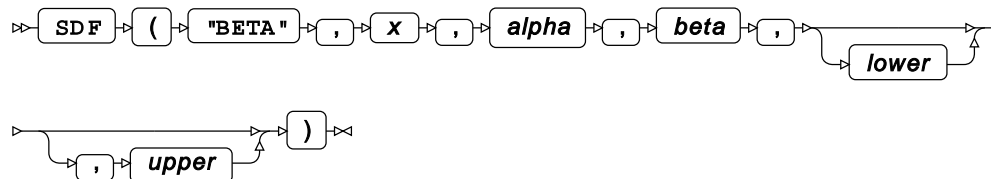
This produces the following output:

```
s1=-0.290977046
s2=-0.290977046
s3=0
s4=M
s5=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last two examples return a missing value because one of the arguments is out of range.

SDF – BETA

Returns the survival of the Beta distribution, based on the shape parameters α and β .



Calculates the survival, or the complement to the cumulative density function, for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*).

The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted. Prior to computation, the value of x is normalised with respect to the domain bounds as follows:

$$z = \frac{x - \text{lower}}{\text{upper} - \text{lower}}$$

This normalised value is then used in further calculations.

This function is defined under the following conditions:

$$\begin{aligned}
 & \text{lower} < \text{upper} \\
 & \begin{cases} \text{if } z \leq 0 & \text{return } 1 \\ \text{if } z \geq 1 & \text{return } 0 \\ \text{otherwise} & \text{return } f(z; \alpha, \beta) \end{cases} \\
 & f(z; \alpha, \beta) = 1 - I_z(\alpha, \beta) = 1 - \frac{B(z; \alpha, \beta)}{B(\alpha, \beta)} \\
 & B(z; \alpha, \beta) = \int_0^z t^{\alpha-1} (1-t)^{\beta-1} dt \\
 & B(\alpha, \beta) = B(z; \alpha, \beta)|_{z=1}
 \end{aligned}$$

where $I_z(\alpha, \beta)$ is the regularised incomplete Beta function; $B(z; \alpha, \beta)$ is the incomplete Beta function; and $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = SDF ("BETA", 0.7, 0.7, 3, -1, 5);  
PUT s1=;  
s2 = SDF ("BETA", 2.7, 0.7, 3, 1, 7);  
PUT s2=;  
s3 = SDF ("BETA", -3, 0.7, 3, -1, 5);  
PUT s3=;  
s4 = SDF ("BETA", 7, 0.7, 3, -1, 5);  
PUT s4=;  
s5 = SDF ("BETA", 0, -0.7, 3, -1, 5);  
PUT s5=;  
RUN;
```

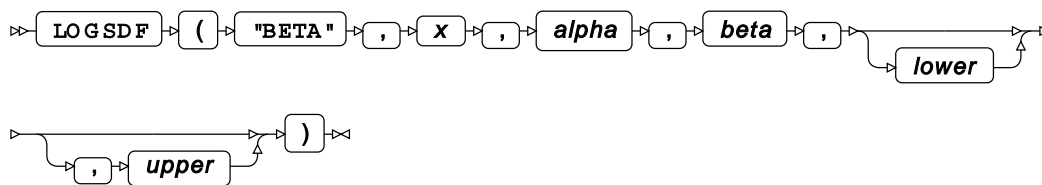
This produces the following output:

```
s1=0.2524671635  
s2=0.2524671635  
s3=1  
s4=0  
s5=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last example returns a missing value because one of the arguments is out of range.

LOGSDF – BETA

Returns the natural logarithm of the survival of the Beta distribution, based on the shape parameters α and β .



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Beta distribution at point x , based on the shape parameters α (*alpha*) and β (*beta*).

The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted. Prior to computation, the value of x is normalised with respect to the domain bounds as follows:

$$z = \frac{x - \text{lower}}{\text{upper} - \text{lower}}$$

This normalised value is then used in further calculations.

This function is defined under the following conditions:

$$\text{lower} < \text{upper}, x < \text{upper}$$

$$\begin{cases} \text{if } z \leq 0 & \text{return } 0 \\ \text{if } 0 < z < 1 & \text{return } f(z; \alpha, \beta) \end{cases}$$

$$f(z; \alpha, \beta) = \log[1 - I_z(\alpha, \beta)] = \log\left[1 - \frac{B(z; \alpha, \beta)}{B(\alpha, \beta)}\right]$$

$$B(z; \alpha, \beta) = \int_0^z t^{\alpha-1} (1-t)^{\beta-1} dt$$

$$B(\alpha, \beta) = B(z; \alpha, \beta)|_{z=1}$$

where $I_z(\alpha, \beta)$ is the regularised incomplete Beta function; $B(z; \alpha, \beta)$ is the incomplete Beta function; and $B(\alpha, \beta)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

Restriction: $x < \text{upper}$

If the argument is out of range, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $\text{upper} > \text{lower}$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF ("BETA", 0.7, 0.7, 3, -1, 5);
  PUT s1=;
  s2 = LOGSDF ("BETA", 2.7, 0.7, 3, 1, 7);
  PUT s2=;
  s3 = LOGSDF ("BETA", -3, 0.7, 3, -1, 5);
  PUT s3=;
  s4 = LOGSDF ("BETA", 7, 0.7, 3, -1, 5);
  PUT s4=;
  s5 = LOGSDF ("BETA", 0, -0.7, 3, -1, 5);
  PUT s5=;
RUN;
```

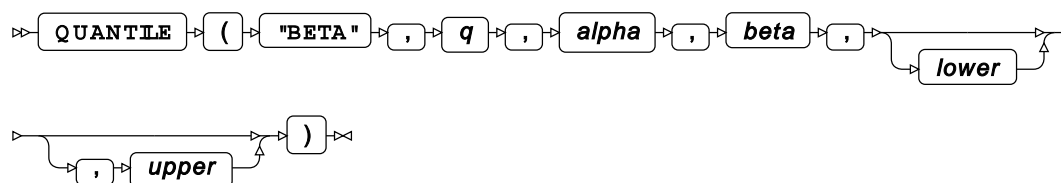
This produces the following output:

```
s1=-1.376474084
s2=-1.376474084
s3=0
s4=M
s5=.
```

The second example demonstrates the effect of normalisation of x with respect to the domain bounds. In the second example the function result remains the same when x , *lower* and *upper* are increased or decreased by the same value. The last two examples return a missing value because one of the arguments is out of range.

QUANTILE – BETA

Returns the quantile of the Beta distribution for a specified probability value, based on the shape parameters α and β . This function is similar to BETAINV where optional arguments are set to default values.



Calculates the quantile x , or the inverse of the cumulative density function, for the Beta distribution for probability value q based on the shape parameters α (*alpha*) and β (*beta*). The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

This function is defined under the following conditions:

$$0 \leq q \leq 1, \alpha > 0, \beta > 0$$
$$\begin{cases} \text{if } q=0 & \text{return } lower \\ \text{if } q=1 & \text{return } upper \\ \text{otherwise} & \text{return } f(q) \end{cases}$$
$$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Beta distribution, see section *CDF – BETA* [↗](#) (page 806).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q \leq 1$

If the argument is out of range or contains a missing value, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $beta > 0$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE ("BETA", 0, 0.7, 3, -1, 5);
  PUT s1=;
  s2 = QUANTILE ("BETA", 1, 0.7, 3, -1, 5);
  PUT s2=;
  s3 = QUANTILE ("BETA", 2, 0.7, 3, -1, 5);
  PUT s3=;
RUN;
```

This produces the following output:

```
s1=-1
s2=5
s3=.
```

The last example returns a missing value because one of the arguments is out of range.

BETAINV

Returns the quantile of the Beta distribution for a specified probability value, based on the shape parameters α and β . This function is similar to QUANTILE – BETA where optional arguments are set to default values.

```
➤ BETAINV ( q , alpha , beta ) ➤
```

Calculates the quantile x , or the inverse of the cumulative density function, for the Beta distribution for probability value q based on the shape parameters α (*alpha*) and β (*beta*).

This function is defined under the following conditions:

$$0 \leq q \leq 1, \alpha > 0, \beta > 0$$
$$\begin{cases} \text{if } q=0 & \text{return } 0 \\ \text{if } q=1 & \text{return } 1 \\ \text{otherwise} & \text{return } f(q) \end{cases}$$
$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Beta distribution where optional arguments are set to their default values, see section [CDF – BETA](#) (page 806).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q \leq 1$

If the argument is out of range or contains a missing value, a missing value is returned.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $beta > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Beta distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = BETAINV (0,0.7,3);  
  PUT s1=;  
  s2 = BETAINV (1,0.7,3);  
  PUT s2=;  
  s3 = BETAINV (2,0.7,3);  
  PUT s3=;  
RUN;
```

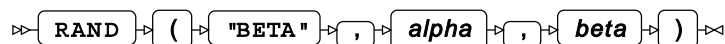
This produces the following output:

```
s1=0  
s2=1  
s3=.
```

The last example returns a missing value because one of the arguments is out of range.

RAND – BETA

Returns a random number from the Beta distribution based on the shape parameters α and β .



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

The return value is between 0 and 1, not including the bounds.

alpha

Type: Numeric

The first shape parameter for the distribution.

Restriction: $\alpha > 0$

If the argument is out of range, a missing value is returned.

beta

Type: Numeric

The second shape parameter for the distribution.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

Example

In this example, a random number from the Beta distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("BETA", 5, 7);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.66276642
0.5280493354
0.6088615973
0.3683775145
0.5415439703
```

Running the DATA step again produces the following output.

```
The random numbers are:
0.247427599
0.4217970341
0.4026729573
0.5006055913
0.5966151994
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(10);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("BETA", 5, 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.7409740272
0.4877923354
0.5533304015
0.8536038268
0.3520169329
```

Running the DATA step again produces the same output.

Binomial distribution

Functions and CALL routines for the Binomial distribution at a specified point, based on the probability of success.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – BINOMIAL [↗](#)..... 825

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Returns the probability density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of PMF – BINOMIAL.

PMF – BINOMIAL [↗](#)..... 826

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Returns the probability mass of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of PDF – BINOMIAL.

LOGPDF – BINOMIAL [↗](#)..... 828

$$\log \left[\binom{n}{k} p^k (1-p)^{n-k} \right]$$

Returns the natural logarithm of the probability density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of LOGPMF – BINOMIAL.

LOGPMF – BINOMIAL [↗](#)..... 830


$$\log \left[\binom{n}{k} p^k (1-p)^{n-k} \right]$$

Returns the natural logarithm of the probability mass of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of LOGPDF – BINOMIAL.

CDF – BINOMIAL [↗](#)..... 831

$$1 - I_p(k+1, n-k)$$

Returns the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is similar to PROBBNML.	
PROBBNML ↗	833
$1 - I_p(k+1, n-k)$	
Returns the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is similar to CDF – BINOMIAL.	
LOGCDF – BINOMIAL ↗	834
$\log[1 - I_p(k+1, n-k)]$	
Returns the natural logarithm of the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.	
SDF – BINOMIAL ↗	836
$I_p(k+1, n-k) = \frac{B(p; k+1, n-k)}{B(k+1, n-k)}$	
Returns the survival of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.	
LOGSDF – BINOMIAL ↗	838
$\log I_p(k+1, n-k)$	
Returns the natural logarithm of the survival of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.	
QUANTILE – BINOMIAL ↗	839
$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$	
Returns the quantile of the Binomial distribution for a specified probability value, based on the probability of success and the number of Bernoulli trials.	
DEVIANC – BINOMIAL ↗	841
$\begin{cases} \text{if } x=n & \text{return } 2x \log \frac{x}{\mu} \\ \text{if } 0 < x < n & \text{return } 2(x \log \frac{x}{\mu} + (n-x) \log \frac{n-x}{n-\mu}) \end{cases}$	
Returns the deviance of the Binomial distribution at a specified point, based on the distribution mean and the number of trials.	
RAND – BINOMIAL ↗	843
Returns a random number from the Binomial distribution based on the probability of success and the the number of Bernoulli trials. This function is similar to RANBIN and CALL RANBIN.	
RANBIN ↗	845
Returns a random number from the Binomial distribution based on the probability of success and the the number of Bernoulli trials. This function is similar to RAND – BINOMIAL and CALL RANBIN.	

CALL RANBIN 848
 Returns a random number from the Binomial distribution based on the probability of success and the number of Bernoulli trials. This routine is similar to function RAND – BINOMIAL and RANBIN.

PDF – BINOMIAL

Returns the probability density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of PMF – BINOMIAL.

➤ PDF ➤ (➤ "BINOMIAL" ➤ , ➤ k ➤ , ➤ p ➤ , ➤ n ➤) ➤

Calculates the probability density function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ k &\in \mathbb{Z} \end{aligned}$$

$$\left\{ \begin{array}{ll} \text{if } k < 0 \text{ or } k > n & \text{return } 0 \\ \text{if } p = 0 \text{ and } k = 0 & \text{return } 1 \\ \text{if } p = 0 \text{ and } k > 0 & \text{return } 0 \\ \text{if } p = 1 \text{ and } k = n & \text{return } 1 \\ \text{if } p = 1 \text{ and } k < n & \text{return } 0 \\ \text{if } 0 < p < 1 \text{ and } 0 < k < n & \text{return } f(p; k, n) \end{array} \right.$$

$$f(p; k, n) = \binom{n}{k} p^k (1-p)^{n-k} \text{ where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Return type: Numeric

k

Type: Numeric

The number of successes.

Restriction: k must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF("BINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = PDF("BINOMIAL", -1, 0.7, 8);
  PUT s2=;
  s3 = PDF("BINOMIAL", 10, 0.7, 8);
  PUT s3=;
  s4 = PDF("BINOMIAL", 0.6, 0.7, 8);
  PUT s4=;
  s5 = PDF("BINOMIAL", 0, -1.7, 8);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.00006561
s2=0
s3=0
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

PMF – BINOMIAL

Returns the probability mass of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of PDF – BINOMIAL.

```
PMF ( "BINOMIAL" , k , p , n )
```

Calculates the probability mass function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned}
 &0 \leq p \leq 1 \\
 &n \geq 0, n \in \mathbb{Z} \\
 &k \in \mathbb{Z} \\
 &\left\{ \begin{array}{ll} \text{if } k < 0 \text{ or } k > n & \text{return } 0 \\ \text{if } p = 0 \text{ and } k = 0 & \text{return } 1 \\ \text{if } p = 0 \text{ and } k > 0 & \text{return } 0 \\ \text{if } p = 1 \text{ and } k = n & \text{return } 1 \\ \text{if } p = 1 \text{ and } k < n & \text{return } 0 \\ \text{if } 0 < p < 1 \text{ and } 0 < k < n & \text{return } f(p; k, n) \end{array} \right. \\
 &f(p; k, n) = \binom{n}{k} p^k (1-p)^{n-k} \text{ where } \binom{n}{k} = \frac{n!}{k!(n-k)!}
 \end{aligned}$$

Return type: Numeric

k

Type: Numeric

The number of successes.

Restriction: *k* must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("BINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = PMF ("BINOMIAL", -1, 0.7, 8);
  PUT s2=;
  s3 = PMF ("BINOMIAL", 10, 0.7, 8);
  PUT s3=;
  s4 = PMF ("BINOMIAL", 0.6, 0.7, 8);
  PUT s4=;
  s5 = PMF ("BINOMIAL", 0, -1.7, 8);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.00006561
s2=0
s3=0
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGPDF – BINOMIAL

Returns the natural logarithm of the probability density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of LOGPMF – BINOMIAL.

LOGPDF ("BINOMIAL" , *k* , *p* , *n*)

Calculates the natural logarithm of the probability density function for the Binomial distribution for the number of successes *k* based on the probability of success *p* and the number of Bernoulli trials *n*.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ 0 &\leq k \leq n, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } p=0 \text{ and } k=0 & \text{return } 0 \\ \text{if } p=1 \text{ and } k=n & \text{return } 0 \\ \text{if } 0 < p < 1 \text{ and } 0 < k < n & \text{return } f(p; k, n) \end{cases}$$

$$f(p; k, n) = \log \left[\binom{n}{k} p^k (1-p)^{n-k} \right] = \log \binom{n}{k} + k \log p + (n-k) \log (1-p)$$

$$\text{where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Return type: Numeric

k

Type: Numeric

The number of successes.

Restriction: $0 \leq k \leq n$ and k must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPDF("BINOMIAL", 0, 0.7, 8);  
  PUT s1=;  
  s2 = LOGPDF("BINOMIAL", -1, 0.7, 8);  
  PUT s2=;  
  s3 = LOGPDF("BINOMIAL", 0, -1.7, 8);  
  PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-9.631782435  
s2=.  
s3=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGPMF – BINOMIAL

Returns the natural logarithm of the probability mass of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is an alias of LOGPDF – BINOMIAL.

LOGPMF ("BINOMIAL" , k , p , n)

Calculates the natural logarithm of the probability mass function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$0 \leq p \leq 1$$

$$n \geq 0, n \in \mathbb{Z}$$

$$0 \leq k \leq n, k \in \mathbb{Z}$$

$$\begin{cases} \text{if } p=0 \text{ and } k=0 & \text{return } 0 \\ \text{if } p=1 \text{ and } k=n & \text{return } 0 \\ \text{if } 0 < p < 1 \text{ and } 0 < k < n & \text{return } f(p; k, n) \end{cases}$$

$$f(p; k, n) = \log \left[\binom{n}{k} p^k (1-p)^{n-k} \right] = \log \binom{n}{k} + k \log p + (n-k) \log (1-p)$$

$$\text{where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Return type: Numeric

k

Type: Numeric

The number of successes.

Restriction: $0 \leq k \leq n$ and k must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF ("BINOMIAL", 0, 0.7, 8);
  PUT s1;
  s2 = LOGPMF ("BINOMIAL", -1, 0.7, 8);
  PUT s2;
  s3 = LOGPMF ("BINOMIAL", 0, -1.7, 8);
  PUT s3;
RUN;
```

This produces the following output:

```
s1=-9.631782435
s2=.
s3=.
```

The last two examples return a missing value because one of the arguments is out of range.

CDF – BINOMIAL

Returns the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is similar to PROBBNML.

⇒ CDF ("BINOMIAL" , k , p , n) ⇒

Calculates the cumulative density function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ k &\in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } k < 0 & \text{return } 0 \\ \text{if } k \geq n & \text{return } 1 \\ \text{otherwise} & \text{return } f(p; k, n) \end{cases}$$

$$f(p; k, n) = 1 - I_p(k+1, n-k) = 1 - \frac{B(p; k+1, n-k)}{B(k+1, n-k)}$$

$$B(p; k+1, n-k) = \int_0^p t^k (1-t)^{n-k-1} dt$$

$$B(k+1, n-k) = B(p; k+1, n-k)|_{p=1}$$

where $I_p(k+1, n-k)$ is the regularised incomplete Beta function; $B(p; k+1, n-k)$ is the incomplete Beta function; and $B(k+1, n-k)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

k

Type: Numeric

The number of successes.

Restriction: k must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = CDF("BINOMIAL", 0, 0.7, 8);  
PUT s1=;  
s2 = CDF("BINOMIAL", -1, 0.7, 8);  
PUT s2=;  
s3 = CDF("BINOMIAL", 10, 0.7, 8);  
PUT s3=;  
s4 = CDF("BINOMIAL", 0.6, 0.7, 8);  
PUT s4=;  
s5 = CDF("BINOMIAL", 0, -1.7, 8);  
PUT s5=;  
RUN;
```

This produces the following output:

```
s1=0.00006561  
s2=0  
s3=1  
s4=.  
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

PROBBNML

Returns the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials. This function is similar to CDF – BINOMIAL.

PROBBNML (p , n , k)

Note:

Function PROBBNML differs from CDF ("BINOMIAL" , k , p , n) in the order of the arguments and in the restrictions imposed on k .

Calculates the cumulative density function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ 0 &\leq k \leq n, k \in \mathbb{Z} \end{aligned}$$

$$f(p; k, n) = 1 - I_p(k+1, n-k) = 1 - \frac{B(p; k+1, n-k)}{B(k+1, n-k)}$$

$$B(p; k+1, n-k) = \int_0^p t^k (1-t)^{n-k-1} dt$$

$$B(k+1, n-k) = B(p; k+1, n-k) \big|_{p=1}$$

where $I_p(k+1, n-k)$ is the regularised incomplete Beta function; $B(p; k+1, n-k)$ is the incomplete Beta function; and $B(k+1, n-k)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: k must be integer and $0 \leq k \leq n$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PROBBNML(0.7,8,0);
  PUT s1=;
  s2 = PROBBNML(0.7,8,-1);
  PUT s2=;
  s3 = PROBBNML(-1.7,8,0);
  PUT s3=;
RUN;
```

This produces the following output:

```
s3=0.00006561
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGCDF – BINOMIAL

Returns the natural logarithm of the cumulative density of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.

```
LOGCDF ( "BINOMIAL" , k , p , n )
```

Calculates the natural logarithm of the cumulative density function for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ k &\geq 0, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } k \geq n & \text{return } 0 \\ \text{if } 0 \leq k \leq n & \text{return } f(p; k, n) \end{cases}$$

$$f(p; k, n) = \log[1 - I_p(k+1, n-k)] = \log\left[1 - \frac{B(p; k+1, n-k)}{B(k+1, n-k)}\right]$$

$$B(p; k+1, n-k) = \int_0^p t^k (1-t)^{n-k-1} dt$$

$$B(k+1, n-k) = B(p; k+1, n-k)|_{p=1}$$

where $I_p(k+1, n-k)$ is the regularised incomplete Beta function; $B(p; k+1, n-k)$ is the incomplete Beta function; and $B(k+1, n-k)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is negative or zero.

k

Type: Numeric

The number of successes.

Restriction: k must be integer and $k \geq 0$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("BINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = LOGCDF ("BINOMIAL", 9, 0.7, 8);
  PUT s2=;
  s3 = LOGCDF ("BINOMIAL", -1, 0.7, 8);
  PUT s3=;
  s4 = LOGCDF ("BINOMIAL", 0, -1.7, 8);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=-1.203972804
s2=0
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

SDF – BINOMIAL

Returns the survival of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.

```
➤ SDF ( "BINOMIAL" , k , p , n ) ➤
```

Calculates the survival, or the complement to the cumulative density function, for the Binomial distribution for the number of successes k based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned}
 &0 \leq p \leq 1 \\
 &n \geq 0, n \in \mathbb{Z} \\
 &k \in \mathbb{Z} \\
 &\begin{cases} \text{if } k < 0 & \text{return } 1 \\ \text{if } k \geq n & \text{return } 0 \\ \text{otherwise} & \text{return } I_p(k+1, n-k) \end{cases} \\
 &I_p(k+1, n-k) = \frac{B(p; k+1, n-k)}{B(k+1, n-k)} \\
 &B(p; k+1, n-k) = \int_0^p t^k (1-t)^{n-k-1} dt \\
 &B(k+1, n-k) = B(p; k+1, n-k) \big|_{p=1}
 \end{aligned}$$

where $I_p(k+1, n-k)$ is the regularised incomplete Beta function; $B(p; k+1, n-k)$ is the incomplete Beta function; and $B(k+1, n-k)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

k

Type: Numeric

The number of successes.

Restriction: k must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
s1 = SDF("BINOMIAL", 0, 0.7, 8);
PUT s1=;
s2 = SDF("BINOMIAL", -1, 0.7, 8);
PUT s2=;
s3 = SDF("BINOMIAL", 10, 0.7, 8);
PUT s3=;
s4 = SDF("BINOMIAL", 0.6, 0.7, 8);
PUT s4=;
s5 = SDF("BINOMIAL", 0, -1.7, 8);
PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.99993439
s2=1
s3=0
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGSDF – BINOMIAL

Returns the natural logarithm of the survival of the Binomial distribution for a specified number of successes based on the probability of success and the number of Bernoulli trials.

➤ LOGSDF ➤ (➤ "BINOMIAL" ➤ , ➤ *k* ➤ , ➤ *p* ➤ , ➤ *n* ➤) ➤

Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Binomial distribution for the number of successes *k* based on the probability of success *p* and the number of Bernoulli trials *n*.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ n &\geq 0, n \in \mathbb{Z} \\ k &< n, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } k < 0 & \text{return } 0 \\ \text{if } 0 \leq k < n & \text{return } f(p; k, n) \end{cases}$$

$$f(p; k, n) = \log I_p(k+1, n-k) = \log B(p; k+1, n-k) - \log B(k+1, n-k)$$

$$B(p; k+1, n-k) = \int_0^p t^k (1-t)^{n-k-1} dt$$

$$B(k+1, n-k) = B(p; k+1, n-k) \Big|_{p=1}$$

where $I_p(k+1, n-k)$ is the regularised incomplete Beta function; $B(p; k+1, n-k)$ is the incomplete Beta function; and $B(k+1, n-k)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is negative or zero.

k

Type: Numeric

The number of successes.

Restriction: $k < n$

If the argument is out of range, a missing value is returned.

p **Type:** Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

 n **Type:** Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF("BINOMIAL", -1, 0.7, 8);
  PUT s1=;
  s2 = LOGSDF("BINOMIAL", 1, 0.7, 8);
  PUT s2=;
  s3 = LOGSDF("BINOMIAL", 10, 0.7, 8);
  PUT s3=;
  s4 = LOGSDF("BINOMIAL", 0, -1.7, 8);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=0
s2=-0.001291163
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

QUANTILE – BINOMIAL

Returns the quantile of the Binomial distribution for a specified probability value, based on the probability of success and the number of Bernoulli trials.

```
⇒ [QUANTILE] ( [ "BINOMIAL" ] , [  $q$  ] , [  $p$  ] , [  $n$  ] ) ⇒
```

Calculates the quantile x , or the inverse of the cumulative density function, for the Binomial distribution for probability value q based on the probability of success p and the number of Bernoulli trials n .

This function is defined under the following conditions:

$$\begin{aligned}0 &\leq q \leq 1 \\0 &\leq p \leq 1 \\n &\geq 0, n \in \mathbb{Z} \\&\begin{cases} \text{if } q=0 & \text{return } 0 \\ \text{if } q=1 & \text{return } 1 \\ \text{otherwise} & \text{return } f(q) \end{cases} \\f(q) &= \inf \{x: q \leq \text{CDF}(x)\}\end{aligned}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Binomial distribution, see section *CDF – BINOMIAL* [↗](#) (page 831).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q \leq 1$

If the argument is out of range or contains a missing value, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE("BINOMIAL", 0.7, 0, 8);
  PUT s1=;
  s2 = QUANTILE("BINOMIAL", 0.7, 0.3, 8);
  PUT s2=;
  s3 = QUANTILE("BINOMIAL", -1, 0.7, 8);
  PUT s3=;
  s4 = QUANTILE("BINOMIAL", 0, -1.7, 8);
  PUT s4=;
RUN;
```

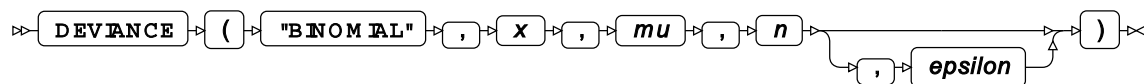
This produces the following output:

```
s1=0
s2=3
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

DEVIANCE – BINOMIAL

Returns the deviance of the Binomial distribution at a specified point, based on the distribution mean and the number of trials.



Calculates the deviance, or goodness of fit, for the generalised linear model of the Binomial distribution at point x based on the distribution mean μ (mu) and the number of trials n , with $n > 0$ and integer, and $0 < x \leq n$. An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The distribution mean is then adjusted so that $n\varepsilon \leq \mu \leq n(1-\varepsilon)$:

$$\begin{cases} \text{if } \mu < n\varepsilon & \text{set } \mu = n\varepsilon \\ \text{if } \mu > n(1-\varepsilon) & \text{set } \mu = n(1-\varepsilon) \end{cases}$$

This adjusted value of μ is used in the subsequent calculation of the deviance.

$$\begin{cases} \text{if } x = n & \text{return } 2x \log \frac{x}{\mu} \\ \text{if } 0 < x < n & \text{return } 2\left(x \log \frac{x}{\mu} + (n-x) \log \frac{n-x}{n-\mu}\right) \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $0 < x \leq n$

If the argument is out of range, a missing value is returned.

mu

Type: Numeric

The distribution mean.

Expected: $0 < \mu < n$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

n

Type: Numeric

The number of trials.

Restriction: $n > 0$

If the argument is out of range, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$

Expected: $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Examples – applying correction to the distribution mean

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = DEVIANCE("BINOMIAL", 0.001, 0.007, 10, 0.0005);
  PUT g1=;
  g2 = DEVIANCE("BINOMIAL", 0.001, 0.007, 10, 0.0010);
  PUT g2=;
  g3 = DEVIANCE("BINOMIAL", 0.001, 0.007, 10, 0.0015);
  PUT g3=;
  g4 = DEVIANCE("BINOMIAL", 0.001, 0.007, 10          );
  PUT g4=;
RUN;
```


This produces the following output:

```
g1=0.0081117815  
g2=0.0134029355  
g3=0.0226035199  
g4=0.0081117815
```

The value of the distribution mean is not corrected in the first example because $\mu > \varepsilon$. However, this condition does not hold in the second and third example, and correction is applied: $\mu = \varepsilon$. This corrected value is used for calculation, yielding different results.

In the fourth example the ε parameter is omitted, so the default value of $\varepsilon = 10^{-12}$ is used. Here, as in the first example, $\mu > \varepsilon$, so no correction is required.

RAND – BINOMIAL

Returns a random number from the Binomial distribution based on the probability of success and the the number of Bernoulli trials. This function is similar to RANBIN and CALL RANBIN.



Each time you execute this function within a **DATA** step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same **DATA** step produce different sequences of random numbers. To initialise the random stream, use **CALL STREAMINIT** [\(page 777\)](#) before this function.

Return type: Numeric

The return value is an integer between 0 and n , inclusive.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 < p < 1$

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n > 0$

If the argument is out of range, a missing value is returned.

Example

In this example, a random number from the Binomial distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("BINOMIAL", 0.75, 10);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
8  
9  
7  
8  
9
```

Running the DATA step again produces the following output.

```
The random numbers are:  
8  
10  
7  
8  
6
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(10);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("BINOMIAL", 0.75, 10);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
10  
6  
7  
9  
9
```

Running the DATA step again produces the same output.

RANBIN

Returns a random number from the Binomial distribution based on the probability of success and the the number of Bernoulli trials. This function is similar to RAND – BINOMIAL and CALL RANBIN.

➤ **RANBIN** ➤ (➤ *seed* ➤ , ➤ *n* ➤ , ➤ *p* ➤) ➤ ➤

The first time you execute this function within a **DATA** step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the **DATA** step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a **DATA** step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is an integer between 0 and *n*, inclusive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

n

Type: Numeric

The number of Bernoulli trials.

Restriction: $n > 0$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 < p < 1$

If the argument is out of range, a missing value is returned.

Basic examples

In this example, a random number from the Binomial distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANBIN(10, 50, 0.2);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
13  
11  
19  
11  
8
```

Running the DATA step again produces the same output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANBIN(0, 50, 0.2);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
11  
9  
11  
10  
10
```

Running the DATA step again produces the following output.

```
The random numbers are:  
6  
9  
12  
5  
8
```

Example — repeated use of seed values

In this example, a random number from the Binomial distribution is returned each time the function is executed. The results are written to the log.

```
DATA _NULL_;
  PUT "The first sequence of random numbers is:";
  result = RANBIN(19, 23, 0.75);
  PUT result;
  result = RANBIN(29, 23, 0.75);
  PUT result;
  result = RANBIN(13, 23, 0.75);
  PUT result;
  result = RANBIN(31, 23, 0.75);
  PUT result;
  result = RANBIN(17, 23, 0.75);
  PUT result;
  result = RANBIN(37, 23, 0.75);
  PUT result;
  result = RANBIN(11, 23, 0.75);
  PUT result;
RUN;

DATA _NULL_;
  PUT "The second sequence of random numbers is:";
  result = RANBIN(19, 23, 0.75);
  PUT result;
  result = RANBIN(97, 23, 0.75);
  PUT result;
  result = RANBIN(37, 23, 0.75);
  PUT result;
  result = RANBIN(41, 23, 0.75);
  PUT result;
  result = RANBIN(71, 23, 0.75);
  PUT result;
  result = RANBIN(67, 23, 0.75);
  PUT result;
  result = RANBIN(51, 23, 0.75);
  PUT result;
RUN;
```

This produces the following output:

```
The first sequence of random numbers is:
17
17
18
20
17
17
16

The second sequence of random numbers is:
17
17
18
20
17
17
16
```

Both **DATA** steps produce the same output because the first seed in each **DATA** step is the same. All subsequent seed values are ignored.

CALL RANBIN

Returns a random number from the Binomial distribution based on the probability of success and the the number of Bernoulli trials. This routine is similar to function **RAND – BINOMIAL** and **RANBIN**.

```
CALL RANBIN ( seed , n , p , x ) ;
```

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a **DATA** step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the **DATA** step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same **DATA** step. To generate a different sequence of random numbers each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is an integer between 0 and *n*, inclusive.

seed**Type:** Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

n**Type:** Numeric

The number of Bernoulli trials.

Restriction: $n > 0$

If the argument is out of range, a missing value is returned.

p**Type:** Numeric

The probability of success in all the trials.

Restriction: $0 < p < 1$

If the argument is out of range, a missing value is returned.

x**Type:** Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Binomial distribution is returned on each iteration of the loop and stored in variable x. The results are written to the log.

This is an example of two different sequences of random numbers generated from the same DATA step.

```
DATA _NULL_;
  PUT "First loop:";
  seed = 19;
  PUT seed=;
  DO i = 1 TO 5;
    CALL RANBIN(seed, 50, 0.2, x);
    PUT x= seed=;
  END;

  PUT "Second loop:";
  seed = 0;
  PUT seed=;
  DO i = 1 TO 5;
    CALL RANBIN(seed, 50, 0.2, x);
    PUT x= seed=;
  END;
RUN;
```

This produces the following output:

```
First loop:
seed=19
x=10 seed=1104426845
x=9 seed=927037761
x=11 seed=1281321492
x=14 seed=1994098043
x=10 seed=1096180836

Second loop:
seed=0
x=5 seed=76099855
x=12 seed=1597485642
x=11 seed=1284594645
x=12 seed=1543772508
x=8 seed=644993642
```

Running the DATA step again produces the following output.

```
First loop:
seed=19
x=10 seed=1104426845
x=9 seed=927037761
x=11 seed=1281321492
x=14 seed=1994098043
x=10 seed=1096180836

Second loop:
seed=0
x=6 seed=115223722
x=12 seed=1558180343
x=10 seed=1046943362
x=11 seed=1489670930
x=9 seed=863372559
```

The first loop produces the same results in both runs of the DATA step because it is initialised with the same non-zero seed value. The second loop produces different sequences of random numbers in the two runs of the DATA step because it is initialised with a zero seed.

Bivariate Normal distribution

Functions for the Bivariate Normal distribution.

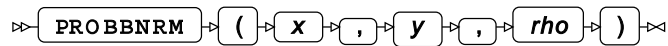
PROBBNRM ↗	851
----------------------------------	-----

$$\frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x \int_{-\infty}^y \exp \left(-\frac{t^2 - 2\rho tu + u^2}{2(1-\rho^2)} \right) du dt$$

Returns the value of the cumulative density function at a specified point for the standard Bivariate Normal distribution where both random variables have mean 0, standard deviation of 1 and the specified correlation coefficient.

PROBBNRM

Returns the value of the cumulative density function at a specified point for the standard Bivariate Normal distribution where both random variables have mean 0, standard deviation of 1 and the specified correlation coefficient.



The bivariate cumulative density function at a point x, y gives the probability that a randomly drawn value from the first distribution is less than or equal to x and a randomly drawn value from the second distribution is less than or equal to y .

This function is defined for $-1 \leq \rho \leq 1$

The calculated value for the standard Bivariate Normal distribution with correlation coefficient ρ (*rho*) is

$$f(x, y; \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x \int_{-\infty}^y \exp\left(-\frac{t^2 - 2\rho tu + u^2}{2(1-\rho^2)}\right) du dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x

Type: Numeric

The point in the first distribution at which to calculate the cumulative density.

y

Type: Numeric

The point in the second distribution at which to calculate the cumulative density.

rho

Type: Numeric

Default: 1

Restriction: $-1 \leq \rho \leq 1$

The correlation coefficient between the distributions.

If the argument is out of range, a missing value is returned.

Basic example

In this example, the value of the cumulative density function for the standard Bivariate Normal distribution is calculated for various values of x , y and ρ . The results are written to the log.

```
DATA _NULL_;  
  s1= PROBBNRM(0, 0, 0);  
  PUT s1=;  
  s2= PROBBNRM(0, 0, 0.5);  
  PUT s2=;  
  s3= PROBBNRM(0, 0, 1);  
  PUT s3=;  
  s4 = PROBBNRM(-0.6, 0.3, 0);  
  PUT s4=;  
  s5 = PROBBNRM(-0.6, 0.3, -0.5);  
  PUT s5=;  
  s6 = PROBBNRM(-0.6, 0.3, -1);  
  PUT s6=;  
RUN;
```

This produces the following output:

```
s1=0.2499999986  
s2=0.333333514  
s3=0.5  
s4=0.1694641404  
s5=0.1008857975  
s6=0
```

The first three examples specify the same pair of points in a standard Bivariate Normal distribution, but with different correlation coefficients. So they return a range of values for the cumulative density function at the specified pair of points.

The fourth, fifth and sixth examples specify a different pair of points in a standard Bivariate Normal distribution, again, with different correlation coefficients. Again, these examples return a range of values for the cumulative density function at the specified pair of points.

Argument errors

In this example, PROBBNRM is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;  
  
  s1 = PROBBNRM(-0.6, 0.3, 1.1);  
  PUT s1=;  
  s2 = PROBBNRM(-0.6, 0.3, -1.1);  
  PUT s2=;  
  
RUN;
```

This produces the following output:

```
s1=.  
s2=.
```

These examples specify invalid values for the correlation coefficient, *rho*. Each generates a message in the log, and returns a missing value.

Cauchy distribution

Functions and `CALL` routines for the Cauchy distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – CAUCHY [↗](#)..... 854

$$\frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

Returns the probability density of the Cauchy distribution based on the location and scale parameters. This function is an alias of PMF – CAUCHY.

PMF – CAUCHY [↗](#)..... 856

$$\frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

Returns the probability mass of the Cauchy distribution based on the location and scale parameters. This function is an alias of PDF – CAUCHY.

LOGPDF – CAUCHY [↗](#)..... 858

$$\log \frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

Returns the natural logarithm of the probability density of the Cauchy distribution based on the location and scale parameters. This function is an alias of LOGPMF – CAUCHY.

LOGPMF – CAUCHY [↗](#)..... 860

$$\log \frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

Returns the natural logarithm of the probability mass of the Cauchy distribution based on the location and scale parameters. This function is an alias of LOGPDF – CAUCHY.

CDF – CAUCHY [↗](#)..... 861

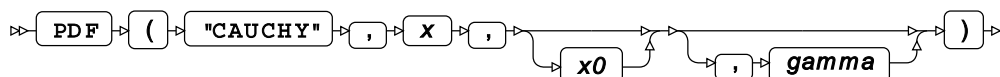
$$\frac{1}{2} + \frac{1}{\pi} \arctan \frac{x-x_0}{\gamma}$$

Returns the cumulative density of the Cauchy distribution based on the location and scale parameters.

LOGCDF – CAUCHY ↗	863
$\log\left(\frac{1}{2} + \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}\right)$	
Returns the natural logarithm of the cumulative density of the Cauchy distribution based on the location and scale parameters.	
SDF – CAUCHY ↗	864
$\frac{1}{2} - \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}$	
Returns the survival of the Cauchy distribution based on the location and scale parameters.	
LOGSDF – CAUCHY ↗	866
$\log\left(\frac{1}{2} - \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}\right)$	
Returns the natural logarithm of the survival of the Cauchy distribution based on the location and scale parameters.	
QUANTILE – CAUCHY ↗	868
$f(q) = x_0 + \gamma \tan\left(x_0\left(q - \frac{1}{2}\right)\right)$	
Returns the quantile of the Cauchy distribution for a specified probability value based on the location and scale parameters.	
RAND – CAUCHY ↗	869
Returns a random number from the Cauchy distribution. This function is similar to RANCAU and CALL RANCAU.	
RANCAU ↗	871
Returns a random number from the Cauchy distribution. This function is similar to RAND – CAUCHY and CALL RANCAU.	
CALL RANCAU ↗	872
Returns a random number from the Cauchy distribution. This routine is similar to function RAND – CAUCHY and RANCAU.	

PDF – CAUCHY

Returns the probability density of the Cauchy distribution based on the location and scale parameters. This function is an alias of PMF – CAUCHY.



Calculates the probability density function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \frac{1}{\pi\gamma \left[1 + \left(\frac{x-x_0}{\gamma} \right)^2 \right]}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

x0

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("CAUCHY", 0.5, -1, 4);
  PUT s1=;
  s2 = PDF ("CAUCHY", 0.5, 0, 1);
  PUT s2=;
  s3 = PDF ("CAUCHY", -0.5, 0, 1);
  PUT s3=;
  s4 = PDF ("CAUCHY", 0.5, -1, -5);
  PUT s4=;
RUN;
```

This produces the following output:

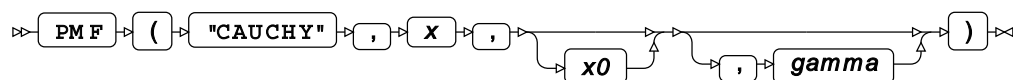
```
s1=0.0697665504
s2=0.2546479089
s3=0.2546479089
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The probability density function of the Cauchy distribution is symmetrical around the location parameter x_0 as illustrated in the second and third examples. The function returns the same result for points which are equidistant from x_0 .

PMF – CAUCHY

Returns the probability mass of the Cauchy distribution based on the location and scale parameters. This function is an alias of PDF – CAUCHY.



Calculates the probability mass function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \frac{1}{\pi \gamma \left[1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right]}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

x0

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = PMF ("CAUCHY", 0.5, -1, 4);  
  PUT s1=;  
  s2 = PMF ("CAUCHY", 0.5, 0, 1);  
  PUT s2=;  
  s3 = PMF ("CAUCHY", -0.5, 0, 1);  
  PUT s3=;  
  s4 = PMF ("CAUCHY", 0.5, -1, -5);  
  PUT s4=;  
RUN;
```

This produces the following output:

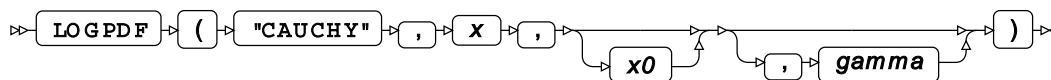
```
s1=0.0697665504
s2=0.2546479089
s3=0.2546479089
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The probability mass function of the Cauchy distribution is symmetrical around the location parameter x_0 as illustrated in the second and third examples. The function returns the same result for points which are equidistant from x_0 .

LOGPDF – CAUCHY

Returns the natural logarithm of the probability density of the Cauchy distribution based on the location and scale parameters. This function is an alias of LOGPMF – CAUCHY.



Calculates the natural logarithm of the probability density function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \log \frac{1}{\pi \gamma \left[1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right]}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

$x0$

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_ ;
  s1 = LOGPDF ("CAUCHY", 0.5, -1, 4) ;
  PUT s1=;
  s2 = LOGPDF ("CAUCHY", 0.5, 0, 1) ;
  PUT s2=;
  s3 = LOGPDF ("CAUCHY", -0.5, 0, 1) ;
  PUT s3=;
  s4 = LOGPDF ("CAUCHY", 0.5, -1, -5) ;
  PUT s4=;
RUN;
```

This produces the following output:

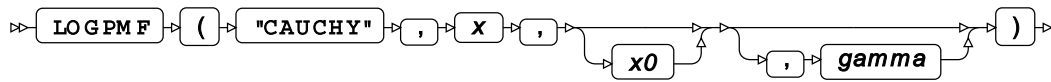
```
s1=-2.546528434
s2=-1.367873437
s3=-1.367873437
s4=.
```

The first three examples show the output when *x* lies within the domain bounds. The fourth example shows the output when *x* falls outside the domain bounds.

The natural logarithm of the probability density function of the Cauchy distribution is symmetrical around the location parameter x_0 as illustrated in the second and third examples. The function returns the same result for points which are equidistant from x_0 .

LOGPMF – CAUCHY

Returns the natural logarithm of the probability mass of the Cauchy distribution based on the location and scale parameters. This function is an alias of LOGPDF – CAUCHY.



Calculates the natural logarithm of the probability mass function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \log \frac{1}{\pi \gamma \left[1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right]}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

$x0$

Optional argument

Type: Numeric

The location parameter.

Default: 0

If $gamma$ is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

$gamma$

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If $gamma$ is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF ("CAUCHY", 0.5, -1, 4);
  PUT s1=;
  s2 = LOGPMF ("CAUCHY", 0.5, 0, 1);
  PUT s2=;
  s3 = LOGPMF ("CAUCHY", -0.5, 0, 1);
  PUT s3=;
  s4 = LOGPMF ("CAUCHY", 0.5, -1, -5);
  PUT s4=;
RUN;
```

This produces the following output:

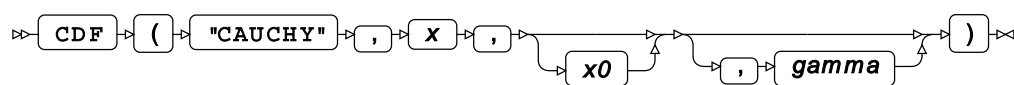
```
s1=-2.546528434
s2=-1.367873437
s3=-1.367873437
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The natural logarithm of the probability mass function of the Cauchy distribution is symmetrical around the location parameter x_0 as illustrated in the second and third examples. The function returns the same result for points which are equidistant from x_0 .

CDF – CAUCHY

Returns the cumulative density of the Cauchy distribution based on the location and scale parameters.



Calculates the cumulative density function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \frac{1}{2} + \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

x0

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("CAUCHY", 3, -1, 4);
  PUT s1=;
  s2 = CDF ("CAUCHY", 0.5, 0, 1);
  PUT s2=;
  s3 = CDF ("CAUCHY", -0.5, 0, 1);
  PUT s3=;
  s4 = CDF ("CAUCHY", 0.5, -1, -5);
  PUT s4=;
RUN;
```

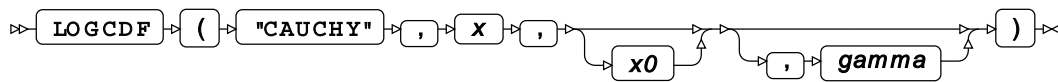
This produces the following output:

```
s1=0.75
s2=0.6475836177
s3=0.3524163823
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

LOGCDF – CAUCHY

Returns the natural logarithm of the cumulative density of the Cauchy distribution based on the location and scale parameters.



Calculates the natural logarithm of the cumulative density function for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \log\left(\frac{1}{2} + \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

$x0$

Optional argument

Type: Numeric

The location parameter.

Default: 0

If $gamma$ is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("CAUCHY", 3, -1, 4);
  PUT s1=;
  s2 = LOGCDF ("CAUCHY", 0.5, 0, 1);
  PUT s2=;
  s3 = LOGCDF ("CAUCHY", -0.5, 0, 1);
  PUT s3=;
  s4 = LOGCDF ("CAUCHY", 0.5, -1, -5);
  PUT s4=;
RUN;
```

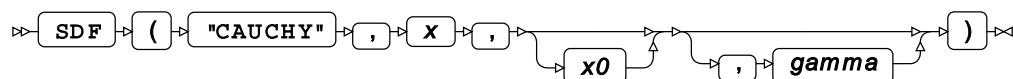
This produces the following output:

```
s1=-0.287682072
s2=-0.434507355
s3=-1.042941898
s4=.
```

The first three examples show the output when *x* lies within the domain bounds. The fourth example shows the output when *x* falls outside the domain bounds.

SDF – CAUCHY

Returns the survival of the Cauchy distribution based on the location and scale parameters.



Calculates the survival, or the complement to the cumulative density function, for the Cauchy distribution at point *x*, based on the location parameter x_0 (*x0*) and the scale parameter γ (*gamma*). Arguments *x0* and *gamma* are optional. If *x0* is omitted, it defaults to 0; if *gamma* is omitted, it defaults to 1.

If $x0$ is specified, then *gamma* must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$
$$f(x; x_0, \gamma) = \frac{1}{2} - \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

x0

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("CAUCHY", 3, -1, 4);
  PUT s1=;
  s2 = SDF ("CAUCHY", 0.5, 0, 1);
  PUT s2=;
  s3 = SDF ("CAUCHY", -0.5, 0, 1);
  PUT s3=;
  s4 = SDF ("CAUCHY", 0.5, -1, -5);
  PUT s4=;
RUN;
```

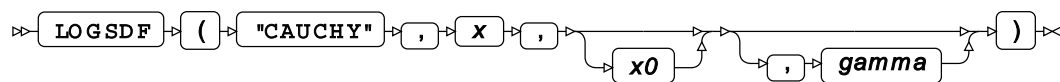
This produces the following output:

```
s1=0.25
s2=0.3524163823
s3=0.6475836177
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

LOGSDF – CAUCHY

Returns the natural logarithm of the survival of the Cauchy distribution based on the location and scale parameters.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Cauchy distribution at point x , based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$). Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$\gamma > 0$$

$$f(x; x_0, \gamma) = \log\left(\frac{1}{2} - \frac{1}{\pi} \arctan \frac{x - x_0}{\gamma}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

x0

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

gamma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGSDF ("CAUCHY", 3, -1, 4);  
  PUT s1=;  
  s2 = LOGSDF ("CAUCHY", 0.5, 0, 1);  
  PUT s2=;  
  s3 = LOGSDF ("CAUCHY", -0.5, 0, 1);  
  PUT s3=;  
  s4 = LOGSDF ("CAUCHY", 0.5, -1, -5);  
  PUT s4=;  
RUN;
```

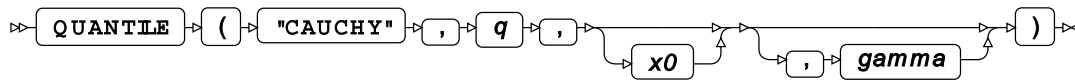
This produces the following output:

```
s1=-1.386294361  
s2=-1.042941898  
s3=-0.434507355  
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

QUANTILE – CAUCHY

Returns the quantile of the Cauchy distribution for a specified probability value based on the location and scale parameters.



Calculates the quantile x , or the inverse of the cumulative density function, for the Cauchy distribution for probability value q based on the location parameter x_0 ($x0$) and the scale parameter γ ($gamma$).

Arguments $x0$ and $gamma$ are optional. If $x0$ is omitted, it defaults to 0; if $gamma$ is omitted, it defaults to 1.

If $x0$ is specified, then $gamma$ must also be specified.

This function is defined under the following conditions:

$$0 < q < 1, \gamma > 0$$

$$f(q) = x_0 + \gamma \tan\left(x_0\left(q - \frac{1}{2}\right)\right)$$

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

$x0$

Optional argument

Type: Numeric

The location parameter.

Default: 0

If $gamma$ is specified, then $x0$ must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

$gamma$

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *gamma* is specified, then *x0* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Cauchy distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE ("CAUCHY", 0.1, -1, 4);
  PUT s1=;
  s2 = QUANTILE ("CAUCHY", 0.9, -1, 4);
  PUT s2=;
  s3 = QUANTILE ("CAUCHY", 0.5, -1, 4);
  PUT s3=;
  s4 = QUANTILE ("CAUCHY", -0.5, -1, 4);
  PUT s4=;
RUN;
```

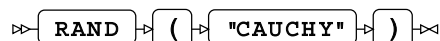
This produces the following output:

```
s1=-13.31073415
s2=11.310734149
s3=-1
s4=.
```

The first three examples show the output when *x* lies within the domain bounds. The fourth example shows the output when *x* falls outside the domain bounds.

RAND – CAUCHY

Returns a random number from the Cauchy distribution. This function is similar to RANCAU and CALL RANCAU.



The distribution is parameterised using a location of 0 and a scale of 1.

This function does not take any variable arguments.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

Example

In this example, a random number from the Cauchy distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("CAUCHY");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.4263776967  
-3.888063075  
1.6837079995  
-1.271628711  
1.042411634
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.855515708  
0.6539476118  
-3.702708893  
0.4270412143  
4.8968502713
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(10);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("CAUCHY");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-0.613114417  
2.4034121057  
-0.649756486  
-1.821516178  
6.856637058
```

Running the DATA step again produces the same output.

RANCAU

Returns a random number from the Cauchy distribution. This function is similar to RAND – CAUCHY and CALL RANCAU.



The distribution is parameterised using a location of 0 and a scale of 1.

The first time you execute this function within a DATA step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the DATA step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the DATA step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a DATA step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, a random number from the Cauchy distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RANCAU(10);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.7404084178
-1.473097832
0.3834782913
-0.012433112
1.1742074908
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the `DATA` step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANCAU(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-19.05625376  
1.2717209805  
5.1594876086  
-0.221519437  
0.7377019906
```

Running the `DATA` step again produces the following output.

```
The random numbers are:-0.511933644  
-5.404156858  
0.8455612831  
-2.727554921  
0.7114644347
```

CALL RANCAU

Returns a random number from the Cauchy distribution. This routine is similar to function `RAND – CAUCHY` and `RANCAU`.

➤ `CALL RANCAU` ➤ `(` ➤ `seed` ➤ `,` ➤ `x` ➤ `)` ➤ `;` ➤

The distribution is parameterised using a location of 0 and a scale of 1.

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a `DATA` step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same `DATA` step. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Cauchy distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  DO i = 1 TO 5;  
    call rancau(10, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
1.7404084178  
-1.473097832  
0.3834782913  
-0.012433112  
1.1742074908
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  DO i = 1 TO 5;  
    call rancau(0, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:
-0.075360642
-0.720706397
1.2159827241
1.80796392
-0.716337281
```

Running the DATA step again produces the following output.

```
The random numbers are:
-0.266269517
-0.77613149
-1.596082366
2.980330735
2.2060086809
```

Chi-Squared distribution

Functions for the Chi-Squared distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

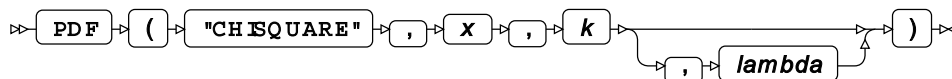
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – CHISQUARE ↗	875
Returns the probability density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of PMF – CHISQUARE.	
PMF – CHISQUARE ↗	876
Returns the probability mass of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of PDF – CHISQUARE.	
LOGPDF – CHISQUARE ↗	876
Returns the natural logarithm of the probability density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of LOGPMF – CHISQUARE.	
LOGPMF – CHISQUARE ↗	877
Returns the natural logarithm of the probability mass of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of LOGPDF – CHISQUARE.	

CDF – CHISQUARE ↗	878
Returns the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to PROBCHI.	
PROBCHI ↗	878
Returns the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to CDF – CHISQUARE.	
LOGCDF – CHISQUARE ↗	879
Returns the natural logarithm of the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.	
SDF – CHISQUARE ↗	879
Returns the survival of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.	
LOGSDF – CHISQUARE ↗	880
Returns the natural logarithm of the survival of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.	
QUANTILE – CHISQUARE ↗	880
Returns the quantile of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to CINV.	
CINV ↗	881
Returns the quantile of the Noncentral Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to QUANTILE – CHISQUARE.	
RAND – CHISQUARE ↗	882
Returns a random number from the Chi-Squared distribution based on the number of degrees of freedom.	

PDF – CHISQUARE

Returns the probability density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of PMF – CHISQUARE.



Return type: Numeric

x

Type: Numeric

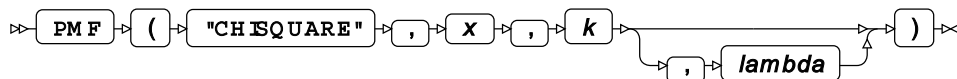
k**Type:** Numeric***lambda***

Optional argument

Type: Numeric

PMF – CHISQUARE

Returns the probability mass of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of PDF – CHISQUARE.

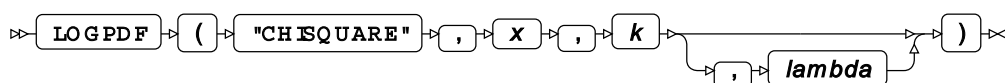
**Return type:** Numeric***x*****Type:** Numeric***k*****Type:** Numeric***lambda***

Optional argument

Type: Numeric

LOGPDF – CHISQUARE

Returns the natural logarithm of the probability density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of LOGPMF – CHISQUARE.



Return type: Numeric

x

Type: Numeric

k

Type: Numeric

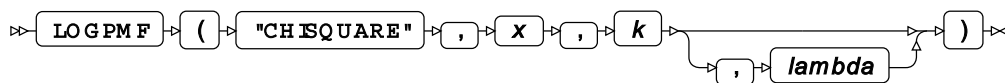
lambda

Optional argument

Type: Numeric

LOGPMF – CHISQUARE

Returns the natural logarithm of the probability mass of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is an alias of LOGPDF – CHISQUARE.



Return type: Numeric

x

Type: Numeric

k

Type: Numeric

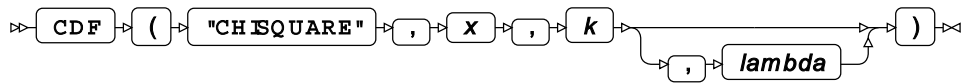
lambda

Optional argument

Type: Numeric

CDF – CHISQUARE

Returns the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to PROBCHI.



Return type: Numeric

x

Type: Numeric

k

Type: Numeric

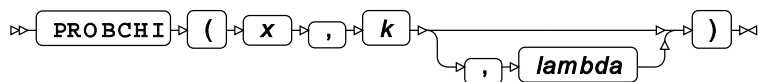
lambda

Optional argument

Type: Numeric

PROBCHI

Returns the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to CDF – CHISQUARE.



Return type: Numeric

x

Type: Numeric

k

Type: Numeric

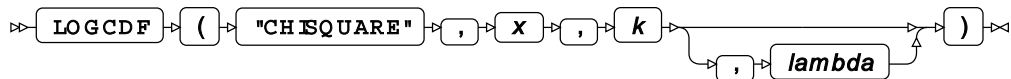
lambda

Optional argument

Type: Numeric

LOGCDF – CHISQUARE

Returns the natural logarithm of the cumulative density of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.



Return type: Numeric

x

Type: Numeric

k

Type: Numeric

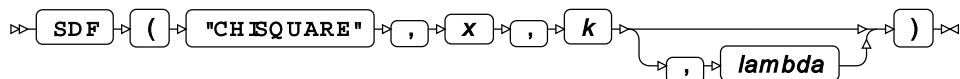
lambda

Optional argument

Type: Numeric

SDF – CHISQUARE

Returns the survival of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.



Return type: Numeric

x

Type: Numeric

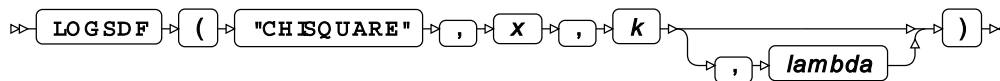
k**Type:** Numeric***lambda***

Optional argument

Type: Numeric

LOGSDF – CHISQUARE

Returns the natural logarithm of the survival of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter.

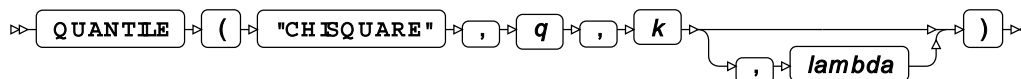
**Return type:** Numeric***x*****Type:** Numeric***k*****Type:** Numeric***lambda***

Optional argument

Type: Numeric

QUANTILE – CHISQUARE

Returns the quantile of the Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to CINV.

**Return type:** Numeric

q

Type: Numeric

k

Type: Numeric

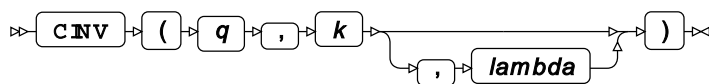
lambda

Optional argument

Type: Numeric

CINV

Returns the quantile of the Noncentral Chi-Squared distribution, based on the number of degrees of freedom and the noncentrality parameter. This function is similar to QUANTILE – CHISQUARE.



Return type: Numeric

q

Type: Numeric

k

Type: Numeric

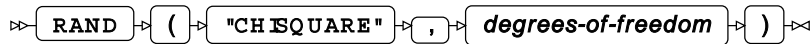
lambda

Optional argument

Type: Numeric

RAND – CHISQUARE

Returns a random number from the Chi-Squared distribution based on the number of degrees of freedom.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

degrees-of-freedom

Type: Numeric

The number of degrees of freedom.

Example

In this example, a random number from the Chi-Squared distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("CHISQUARE", 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.1890726701
2.6570160479
2.7255796359
1.5478911372
1.2643875905
```


Running the DATA step again produces the following output.

```
The random numbers are:
4.1108027773
6.1036866288
2.4825035135
1.5245407666
1.1249219054
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(50);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("CHISQUARE", 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.3922968316
0.6089642297
4.8476385246
0.5618948164
4.6102139672
```

Running the DATA step again produces the same output.

Erlang distribution

Functions for the Erlang distribution.

RAND – ERLANG [↗](#).....883
Returns a random number from the Erlang distribution based on the mean.

RAND – ERLANG

Returns a random number from the Erlang distribution based on the mean.

```
➡ RAND ( "ERLANG " , shape ) ➡
```

The distribution is parameterised using a rate of 1.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS)* - Special issue on uniform random number generation 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

shape

Type: Numeric

The shape of the distribution.

Example

In this example, a random number from the Erlang distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("ERLANG", 3);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
4.0300435051  
6.6808226796  
9.4384477997  
4.2859111604  
4.9152869497
```

Running the `DATA` step again produces the following output.

```
The random numbers are:  
4.6183065003  
5.2119448149  
6.2187519718  
5.5032738869  
9.3680427968
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(50);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("ERLANG", 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
3.2303700361
0.9112274081
2.8616064449
2.3220704166
3.1615075339
```

Running the `DATA` step again produces the same output.

Exponential distribution

Functions and `CALL` routines for the Exponential distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – EXPONENTIAL [↗](#)..... 887

$$\frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right)$$

Returns the probability density of the Exponential distribution, based on the scale parameter. This function is an alias of PMF – EXPONENTIAL.

PMF – EXPONENTIAL [↗](#)..... 888

$$\frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right)$$

Returns the probability mass of the Exponential distribution, based on the scale parameter. This function is an alias of PDF – EXPONENTIAL.

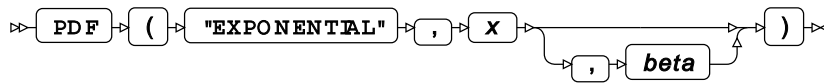
LOGPDF – EXPONENTIAL [↗](#)..... 890

$$-\frac{x}{\beta} + \log \frac{1}{\beta}$$

Returns the natural logarithm of the probability density of the Exponential distribution, based on the scale parameter. This function is an alias of LOGPMF – EXPONENTIAL.	
LOGPMF – EXPONENTIAL ↗	891
$-\frac{x}{\beta} + \log \frac{1}{\beta}$	
Returns the natural logarithm of the probability mass of the Exponential distribution, based on the scale parameter. This function is an alias of LOGPDF – EXPONENTIAL.	
CDF – EXPONENTIAL ↗	893
$1 - \exp\left(-\frac{x}{\beta}\right)$	
Returns the cumulative density of the Exponential distribution, based on the scale parameter.	
LOGCDF – EXPONENTIAL ↗	894
$\log\left[1 - \exp\left(-\frac{x}{\beta}\right)\right]$	
Returns the natural logarithm of the cumulative density of the Exponential distribution, based on the scale parameter.	
SDF – EXPONENTIAL ↗	896
$\exp\left(-\frac{x}{\beta}\right)$	
Returns the survival of the Exponential distribution, based on the scale parameter.	
LOGSDF – EXPONENTIAL ↗	897
$-\frac{x}{\beta}$	
Returns the natural logarithm of the survival of the Exponential distribution, based on the scale parameter.	
QUANTILE – EXPONENTIAL ↗	899
$f(q; \beta) = -\beta \log(1 - q)$	
Returns the quantile of the Exponential distribution, based on the scale parameter.	
RAND – EXPONENTIAL ↗	900
Returns a random number from the Exponential distribution. This function is similar to RANEXP and CALL RANEXP.	
RANEXP ↗	902
Returns a random number from the Exponential distribution. This function is similar to RAND – EXPONENTIAL and CALL RANEXP.	
CALL RANEXP ↗	903
Returns a random number from the Exponential distribution. This routine is similar to function RAND – EXPONENTIAL and RANEXP.	

PDF – EXPONENTIAL

Returns the probability density of the Exponential distribution, based on the scale parameter. This function is an alias of PMF – EXPONENTIAL.



Calculates the probability density function for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\beta > 0$$

$$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \beta) \end{cases}$$

$$f(x; \beta) = \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("EXPONENTIAL", 0.7, 4);
  PUT s1=;
  s2 = PDF ("EXPONENTIAL", 0.7, 2);
  PUT s2=;
  s3 = PDF ("EXPONENTIAL", 0.1, 2);
  PUT s3=;
  s4 = PDF ("EXPONENTIAL", -3, 4);
  PUT s4=;
  s5 = PDF ("EXPONENTIAL", 7, 0);
  PUT s5=;
RUN;
```

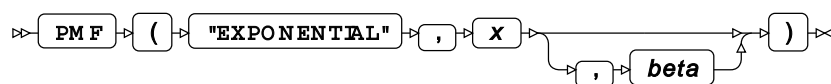
This produces the following output:

```
s1=0.2098642552
s2=0.3523440449
s3=0.4756147123
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

PMF – EXPONENTIAL

Returns the probability mass of the Exponential distribution, based on the scale parameter. This function is an alias of PDF – EXPONENTIAL.



Calculates the probability mass function for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\beta > 0$$

$$\begin{cases} \text{if } x < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \beta) \end{cases}$$

$$f(x; \beta) = \frac{1}{\beta} \exp\left(-\frac{x}{\beta}\right)$$

Return type: Numeric

x**Type:** Numeric

The point at which to calculate the probability mass.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1**Restriction:** must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_ ;
  s1 = PMF ("EXPONENTIAL", 0.7, 4) ;
  PUT s1=;
  s2 = PMF ("EXPONENTIAL", 0.7, 2) ;
  PUT s2=;
  s3 = PMF ("EXPONENTIAL", 0.1, 2) ;
  PUT s3=;
  s4 = PMF ("EXPONENTIAL", -3, 4) ;
  PUT s4=;
  s5 = PMF ("EXPONENTIAL", 7, 0) ;
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.2098642552
s2=0.3523440449
s3=0.4756147123
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

Examples

In these examples, the natural logarithm of the probability density of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("EXPONENTIAL", 0.7, 4);
  PUT s1=;
  s2 = LOGPDF ("EXPONENTIAL", 0.7, 2);
  PUT s2=;
  s3 = LOGPDF ("EXPONENTIAL", 0.1, 2);
  PUT s3=;
  s4 = LOGPDF ("EXPONENTIAL", -3, 4);
  PUT s4=;
  s5 = LOGPDF ("EXPONENTIAL", 7, 0);
  PUT s5=;
RUN;
```

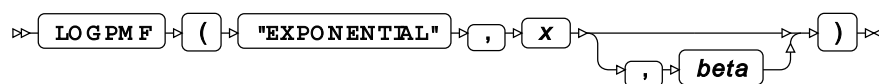
This produces the following output:

```
s1=-1.561294361
s2=-1.043147181
s3=-0.743147181
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGPMF – EXPONENTIAL

Returns the natural logarithm of the probability mass of the Exponential distribution, based on the scale parameter. This function is an alias of LOGPDF – EXPONENTIAL.



Calculates the natural logarithm of the probability mass function for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$x \geq 0, \beta > 0$$

$$f(x; \beta) = -\frac{x}{\beta} + \log \frac{1}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

Restriction: must be positive or zero

If the argument is out of range, a missing value is returned.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_ ;
  s1 = LOGPMF ("EXPONENTIAL", 0.7, 4) ;
  PUT s1=;
  s2 = LOGPMF ("EXPONENTIAL", 0.7, 2) ;
  PUT s2=;
  s3 = LOGPMF ("EXPONENTIAL", 0.1, 2) ;
  PUT s3=;
  s4 = LOGPMF ("EXPONENTIAL", -3, 4) ;
  PUT s4=;
  s5 = LOGPMF ("EXPONENTIAL", 7, 0) ;
  PUT s5=;
RUN;
```

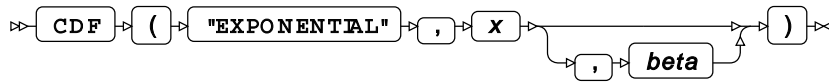
This produces the following output:

```
s1=-1.561294361
s2=-1.043147181
s3=-0.743147181
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

CDF – EXPONENTIAL

Returns the cumulative density of the Exponential distribution, based on the scale parameter.



Calculates the cumulative density function for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\beta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \beta) \end{cases}$$

$$f(x; \beta) = 1 - \exp\left(-\frac{x}{\beta}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("EXPONENTIAL", 0.7, 4);
  PUT s1=;
  s2 = CDF ("EXPONENTIAL", 0.7, 2);
  PUT s2=;
  s3 = CDF ("EXPONENTIAL", 0.1, 2);
  PUT s3=;
  s4 = CDF ("EXPONENTIAL", -3, 4);
  PUT s4=;
  s5 = CDF ("EXPONENTIAL", 7, 0);
  PUT s5=;
RUN;
```

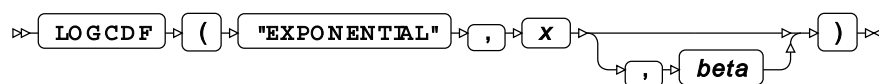
This produces the following output:

```
s1=0.1605429792
s2=0.2953119103
s3=0.0487705755
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGCDF – EXPONENTIAL

Returns the natural logarithm of the cumulative density of the Exponential distribution, based on the scale parameter.



Calculates the natural logarithm of the cumulative density function for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$x > 0, \beta > 0$$

$$f(x; \beta) = \log[1 - \exp(-\frac{x}{\beta})]$$

Return type: Numeric

x**Type:** Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1**Restriction:** must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_ ;
  s1 = LOGCDF ("EXPONENTIAL", 0.7, 4) ;
  PUT s1=;
  s2 = LOGCDF ("EXPONENTIAL", 0.7, 2) ;
  PUT s2=;
  s3 = LOGCDF ("EXPONENTIAL", 0.1, 2) ;
  PUT s3=;
  s4 = LOGCDF ("EXPONENTIAL", -3, 4) ;
  PUT s4=;
  s5 = LOGCDF ("EXPONENTIAL", 7, 0) ;
  PUT s5=;
RUN;
```

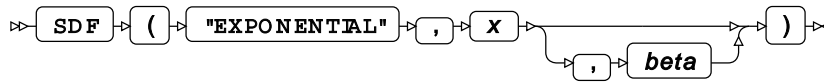
This produces the following output:

```
s1=-1.829193589
s2=-1.219723158
s3=-3.020628109
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

SDF – EXPONENTIAL

Returns the survival of the Exponential distribution, based on the scale parameter.



Calculates the survival, or the complement to the cumulative density function, for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\beta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 1 \\ \text{otherwise} & \text{return } f(x; \beta) \end{cases}$$

$$f(x; \beta) = \exp\left(-\frac{x}{\beta}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("EXPONENTIAL", 0.7, 4);
  PUT s1=;
  s2 = SDF ("EXPONENTIAL", 0.7, 2);
  PUT s2=;
  s3 = SDF ("EXPONENTIAL", 0.1, 2);
  PUT s3=;
  s4 = SDF ("EXPONENTIAL", -3, 4);
  PUT s4=;
  s5 = SDF ("EXPONENTIAL", 7, 0);
  PUT s5=;
RUN;
```

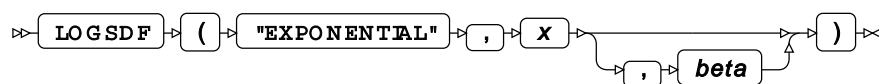
This produces the following output:

```
s1=0.8394570208
s2=0.7046880897
s3=0.9512294245
s4=1
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGSDF – EXPONENTIAL

Returns the natural logarithm of the survival of the Exponential distribution, based on the scale parameter.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Exponential distribution at point x , based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\beta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \theta) \end{cases}$$

$$f(x; \beta) = -\frac{x}{\beta}$$

Return type: Numeric

x **Type:** Numeric

The point at which to calculate the natural logarithm of the survival.

 β

Optional argument

Type: Numeric

The scale parameter.

Default: 1**Restriction:** must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGSDF ("EXPONENTIAL", 0.7, 4);  
  PUT s1=;  
  s2 = LOGSDF ("EXPONENTIAL", 0.7, 2);  
  PUT s2=;  
  s3 = LOGSDF ("EXPONENTIAL", 0.1, 2);  
  PUT s3=;  
  s4 = LOGSDF ("EXPONENTIAL", -3, 4);  
  PUT s4=;  
  s5 = LOGSDF ("EXPONENTIAL", 7, 0);  
  PUT s5=;  
RUN;
```

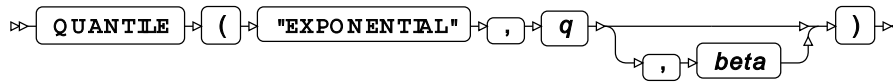
This produces the following output:

```
s1=-0.175  
s2=-0.35  
s3=-0.05  
s4=0  
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

QUANTILE – EXPONENTIAL

Returns the quantile of the Exponential distribution, based on the scale parameter.



Calculates the quantile x , or the inverse of the cumulative density function, for the Exponential distribution for probability value q based on scale parameter β (*beta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$0 < q < 1, \beta > 0$$

$$f(q; \beta) = -\beta \log(1 - q)$$

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

beta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Exponential distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = QUANTILE ("EXPONENTIAL", 0.7, 4);  
  PUT s1=;  
  s2 = QUANTILE ("EXPONENTIAL", 0.7, 2);  
  PUT s2=;  
  s3 = QUANTILE ("EXPONENTIAL", 0.1, 2);  
  PUT s3=;  
  s4 = QUANTILE ("EXPONENTIAL", 0, 4);  
  PUT s4=;  
  s5 = QUANTILE ("EXPONENTIAL", 0.7, 0);  
  PUT s5=;  
RUN;
```

This produces the following output:

```
s1=4.8158912173  
s2=2.4079456087  
s3=0.2107210313  
s4=.  
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output when q equals to zero. The fifth example shows the output when an argument value falls outside the domain bounds.

RAND – EXPONENTIAL

Returns a random number from the Exponential distribution. This function is similar to RANEXP and CALL RANEXP.

→ **RAND** → (→ "EXPONENTIAL" →) →

The distribution is parameterised using a mean of 1.

This function does not take any variable arguments.

Each time you execute this function within a **DATA** step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same **DATA** step produce different sequences of random numbers. To initialise the random stream, use **CALL STREAMINIT** [↗](#) (page 777) before this function.

Return type: Numeric

The return value is positive.

Example

In this example, a random number from the Exponential distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("EXPONENTIAL");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
2.7082011391  
2.3898074454  
0.3126476527  
0.2865067564  
0.1319622419
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.970507831  
2.6047812803  
0.681884276  
0.3436686056  
1.1060946519
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(16);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("EXPONENTIAL");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.2862521114  
0.0710885532  
0.0574738376  
2.8483074967  
1.52861687
```

Running the DATA step again produces the same output.

RANEXP

Returns a random number from the Exponential distribution. This function is similar to RAND – EXPONENTIAL and CALL RANEXP.



The distribution is parameterised using a mean of 1.

The first time you execute this function within a DATA step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the DATA step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the DATA step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a DATA step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is positive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, a random number from the Exponential distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RANEXP(10);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.8641189282
0.1051016758
0.4993573256
0.2530488276
0.3939087162
```

Running the `DATA` step again produces the following output.

If the initial seed is set to zero, each run of the `DATA` step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANEXP(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.4323774084  
0.1566195304  
1.3868878749  
0.7776727044  
0.4901959216
```

Running the `DATA` step again produces the following output.

```
The random numbers are:  
0.633166558  
0.9361323397  
0.4423314352  
1.129363563  
1.3409875818
```

CALL RANEXP

Returns a random number from the Exponential distribution. This routine is similar to function `RAND – EXPONENTIAL` and `RANEXP`.

→ **CALL RANEXP** → (→ *seed* → , → *x* →) → ; →

The distribution is parameterised using a mean of 1.

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a `DATA` step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same `DATA` step. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is positive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Exponential distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    call ranexp(10, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
0.1629593815  
0.3554083799  
0.0017584843  
0.520898765  
1.5323575328
```

Running the `DATA` step again produces the following output.

If the initial seed is set to zero, each run of the `DATA` step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    call ranexp(0, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:
4.3744549453
0.3787492277
0.6879483842
2.2348845064
0.693289297
```

Running the DATA step again produces the following output.

```
The random numbers are:
0.9833834191
0.3000878672
0.1583015061
0.1095265191
1.6490832095
```

Fisher distribution

Functions for the Fisher distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

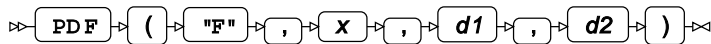
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – F ↗	906
Returns the probability density of the Fisher distribution. This function is an alias of PMF – F.	
PMF – F ↗	906
Returns the probability mass of the Fisher distribution. This function is an alias of PDF – F.	
LOGPDF – F ↗	907
Returns the natural logarithm of the probability density of the Fisher distribution. This function is an alias of LOGPMF – F.	
LOGPMF – F ↗	907
Returns the natural logarithm of the probability mass of the Fisher distribution. This function is an alias of LOGPDF – F.	
CDF – F ↗	908
Returns the cumulative density of the Fisher distribution.	
PROBF ↗	908
LOGCDF – F ↗	909
Returns the natural logarithm of the cumulative density of the Fisher distribution.	

SDF – F ↗	910
Returns the survival of the Fisher distribution.	
LOGSDF – F ↗	910
Returns the natural logarithm of the survival of the Fisher distribution.	
QUANTILE – F ↗	911
Returns the quantile of the Fisher distribution.	
FINV ↗	911
RAND – F ↗	912
Returns a random number from the Fisher distribution based on the degrees of freedom.	

PDF – F

Returns the probability density of the Fisher distribution. This function is an alias of PMF – F.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

d1

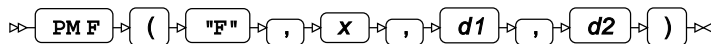
Type: Numeric

d2

Type: Numeric

PMF – F

Returns the probability mass of the Fisher distribution. This function is an alias of PDF – F.



Return type: Numeric

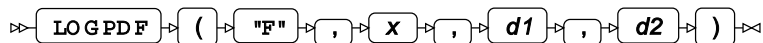
x**Type:** Numeric

The point at which to calculate the probability mass.

d1**Type:** Numeric**d2****Type:** Numeric

LOGPDF – F

Returns the natural logarithm of the probability density of the Fisher distribution. This function is an alias of LOGPMF – F.

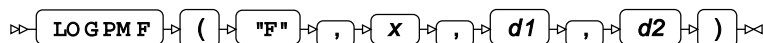
**Return type:** Numeric**x****Type:** Numeric

The point at which to calculate the natural logarithm of the probability density.

d1**Type:** Numeric**d2****Type:** Numeric

LOGPMF – F

Returns the natural logarithm of the probability mass of the Fisher distribution. This function is an alias of LOGPDF – F.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

d1

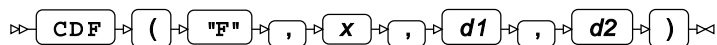
Type: Numeric

d2

Type: Numeric

CDF – F

Returns the cumulative density of the Fisher distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

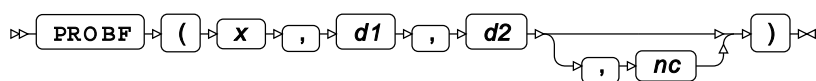
d1

Type: Numeric

d2

Type: Numeric

PROBF



Return type: Numeric

x

Type: Numeric

d1

Type: Numeric

d2

Type: Numeric

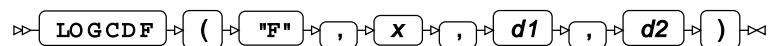
nc

Optional argument

Type: Numeric

LOGCDF – F

Returns the natural logarithm of the cumulative density of the Fisher distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

d1

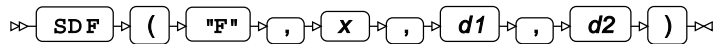
Type: Numeric

d2

Type: Numeric

SDF – F

Returns the survival of the Fisher distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

d1

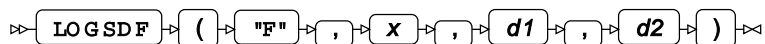
Type: Numeric

d2

Type: Numeric

LOGSDF – F

Returns the natural logarithm of the survival of the Fisher distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

d1

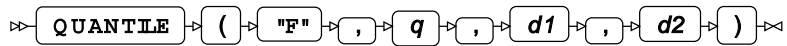
Type: Numeric

d2

Type: Numeric

QUANTILE – F

Returns the quantile of the Fisher distribution.



Return type: Numeric

q

Type: Numeric

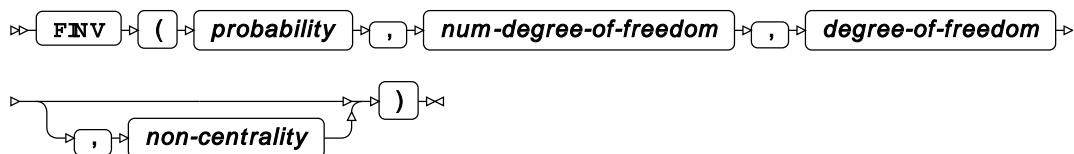
d1

Type: Numeric

d2

Type: Numeric

FINV



Return type: Numeric

probability

Type: Numeric

num-degree-of-freedom

Type: Numeric

degree-of-freedom

Type: Numeric

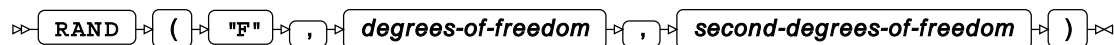
non-centrality

Optional argument

Type: Numeric

RAND – F

Returns a random number from the Fisher distribution based on the degrees of freedom.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

degrees-of-freedom**Type:** Numeric

The first number of degrees of freedom for the distribution.

second-degrees-of-freedom**Type:** Numeric

The second number of degrees of freedom for the distribution.

Example

In this example, a random number from the Fisher distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("F", 7,3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.5580202043
0.2275444934
2.9556306842
0.2816658411
0.9441069678
```

Running the DATA step again produces the following output.

```
The random numbers are:
0.436236944
0.4925625125
1.6689190328
5.3555436202
1.1726657119
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(16);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("F", 7,3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.3151390248
1.3442231108
0.457142241
2.3632063261
1.8960485237
```

Running the DATA step again produces the same output.

Gamma distribution

Functions and `CALL` routines for the Gamma distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

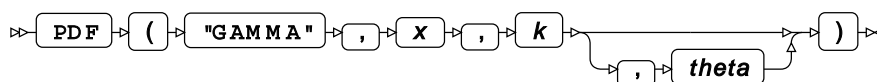
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – GAMMA ↗	915
$\frac{x^{k-1} \exp\left(-\frac{x}{\theta}\right)}{\theta^k \Gamma(k)}$	
Returns the probability density of the Gamma distribution, based on the shape and scale parameters. This function is an alias of PMF – GAMMA.	
PMF – GAMMA ↗	917
$\frac{x^{k-1} \exp\left(-\frac{x}{\theta}\right)}{\theta^k \Gamma(k)}$	
Returns the probability mass of the Gamma distribution, based on the shape and scale parameters. This function is an alias of PDF – GAMMA.	
LOGPDF – GAMMA ↗	919
$-\frac{x}{\theta} + \log \frac{x^{k-1}}{\theta^k \Gamma(k)}$	
Returns the natural logarithm of the probability density of the Gamma distribution, based on the shape and scale parameters. This function is an alias of LOGPMF – GAMMA.	
LOGPMF – GAMMA ↗	921
$-\frac{x}{\theta} + \log \frac{x^{k-1}}{\theta^k \Gamma(k)}$	
Returns the natural logarithm of the probability mass of the Gamma distribution, based on the shape and scale parameters. This function is an alias of LOGPDF – GAMMA.	
CDF – GAMMA ↗	923
$\frac{\gamma\left(k, \frac{x}{\theta}\right)}{\Gamma(k)}$	
Returns the cumulative density of the Gamma distribution, based on the shape and scale parameters. This function is similar to PROBGAM where the optional argument in CDF – GAMMA is set to its default value.	
PROBGAM ↗	925
$\frac{\gamma(k, x)}{\Gamma(k)}$	
Returns the cumulative density of the Gamma distribution, based on the shape and scale parameters. This function is similar to CDF – GAMMA where the optional argument in CDF – GAMMA is set to its default value.	
LOGCDF – GAMMA ↗	927
$\log \frac{\gamma\left(k, \frac{x}{\theta}\right)}{\Gamma(k)}$	
Returns the natural logarithm of the cumulative density of the Gamma distribution, based on the shape and scale parameters.	
SDF – GAMMA ↗	929
$1 - \frac{\gamma\left(k, \frac{x}{\theta}\right)}{\Gamma(k)}$	
Returns the survival of the Gamma distribution, based on the shape and scale parameters.	

LOGSDF – GAMMA ↗	930
$\log \left[1 - \frac{\gamma \left(k, \frac{x}{\theta} \right)}{\Gamma(k)} \right]$	
Returns the natural logarithm of the survival of the Gamma distribution, based on the shape and scale parameters.	
QUANTILE – GAMMA ↗	932
$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$	
Returns the quantile of the Gamma distribution, based on the shape and scale parameters. This function is similar to GAMINV where the optional argument in QUANTILE – GAMMA is set to its default value.	
GAMINV ↗	934
$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$	
Returns the quantile of the Gamma distribution, based on the shape and scale parameters. This function is similar to QUANTILE – GAMMA where the optional argument in QUANTILE – GAMMA is set to its default value.	
DEVIANC – GAMMA ↗	935
$2 \left(\frac{x - \mu}{\mu} - \log \frac{x}{\mu} \right)$	
Returns the deviance of the Gamma distribution at a specified point, based on the distribution mean.	
RAND – GAMMA ↗	938
Returns a random number from the Gamma distribution based on the shape. This function is similar to RANGAM and CALL RANGAM.	
RANGAM ↗	939
Returns a random number from the Gamma distribution based on the shape. This function is similar to RAND – GAMMA and CALL RANGAM.	
CALL RANGAM ↗	941
Returns a random number from the Gamma distribution based on the shape. This routine is similar to function RAND – GAMMA and RANGAM.	

PDF – GAMMA

Returns the probability density of the Gamma distribution, based on the shape and scale parameters. This function is an alias of PMF – GAMMA.



Calculates the probability density function for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\theta > 0$$

$$\begin{cases} \text{if } x=0 \text{ then } k \geq 1 \\ \text{otherwise } k > 0 \end{cases}$$

$$\begin{cases} \text{if } x < 0 \text{ and } k > 0 & \text{return } 0 \\ \text{if } x = 0 \text{ and } k > 1 & \text{return } 0 \\ \text{if } x = 0 \text{ and } k = 1 & \text{return } 1/\theta \\ \text{if } x > 0 \text{ and } k > 0 & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = \frac{x^{k-1} \exp\left(-\frac{x}{\theta}\right)}{\theta^k \Gamma(k)}$$

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

where $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

k

Type: Numeric

The shape parameter.

Restriction: if $x = 0$ then k must be greater than or equal to 1; for all other values of x , k must be greater than 0

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = PDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = PDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = PDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = PDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

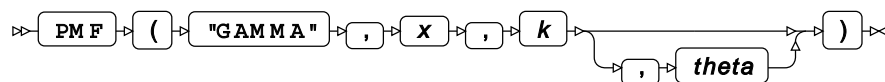
This produces the following output:

```
s1=0.0367262447
s2=0.3476097127
s3=0.0719556586
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

PMF – GAMMA

Returns the probability mass of the Gamma distribution, based on the shape and scale parameters. This function is an alias of PDF – GAMMA.



Calculates the probability mass for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$\theta > 0$$

$$\begin{cases} \text{if } x=0 \text{ then } k \geq 1 \\ \text{otherwise } k > 0 \end{cases}$$

$$\begin{cases} \text{if } x < 0 \text{ and } k > 0 & \text{return } 0 \\ \text{if } x = 0 \text{ and } k > 1 & \text{return } 0 \\ \text{if } x = 0 \text{ and } k = 1 & \text{return } 1/\theta \\ \text{if } x > 0 \text{ and } k > 0 & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = \frac{x^{k-1} \exp\left(-\frac{x}{\theta}\right)}{\theta^k \Gamma(k)}$$

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

where $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

k

Type: Numeric

The shape parameter.

Restriction: if $x = 0$ then k must be greater than or equal to 1; for all other values of x , k must be greater than 0

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = PMF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = PMF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = PMF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = PMF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

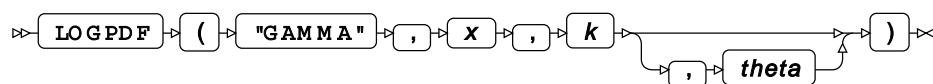
This produces the following output:

```
s1=0.0367262447
s2=0.3476097127
s3=0.0719556586
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGPDF – GAMMA

Returns the natural logarithm of the probability density of the Gamma distribution, based on the shape and scale parameters. This function is an alias of LOGPMF – GAMMA.



Calculates the natural logarithm of the probability density for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$x \geq 0, \theta > 0$$

$$\begin{cases} \text{if } x=0 \text{ then } k=1 \\ \text{otherwise } k>0 \end{cases}$$

$$\begin{cases} \text{if } x=0 \text{ and } k=1 & \text{return } \log(1/\theta) \\ \text{if } x>0 \text{ and } k>0 & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = -\frac{x}{\theta} + \log \frac{x^{k-1}}{\theta^k \Gamma(k)}$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

Restriction: must be positive or zero

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The shape parameter.

Restriction: if $x = 0$ then k must be equal to 1; for all other values of x , k must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = LOGPDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = LOGPDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = LOGPDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = LOGPDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

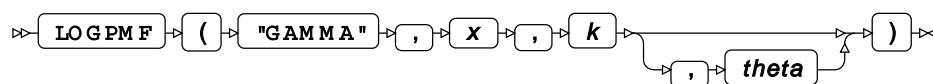
This produces the following output:

```
s1=-3.304263666
s2=-1.056674944
s3=-2.631705202
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGPMF – GAMMA

Returns the natural logarithm of the probability mass of the Gamma distribution, based on the shape and scale parameters. This function is an alias of LOGPDF – GAMMA.



Calculates the natural logarithm of the probability mass for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$x \geq 0, \theta > 0$$

$$\begin{cases} \text{if } x=0 \text{ then } k=1 \\ \text{otherwise } k>0 \end{cases}$$

$$\begin{cases} \text{if } x=0 \text{ and } k=1 & \text{return } \log(1/\theta) \\ \text{if } x>0 \text{ and } k>0 & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = -\frac{x}{\theta} + \log \frac{x^{k-1}}{\theta^k \Gamma(k)}$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

Restriction: must be positive or zero

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The shape parameter.

Restriction: if $x = 0$ then k must be equal to 1; for all other values of x , k must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = LOGPMF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = LOGPMF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = LOGPMF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = LOGPMF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

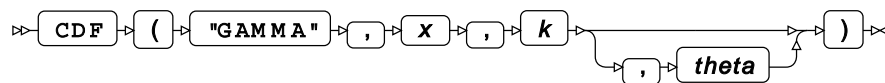
This produces the following output:

```
s1=-3.304263666
s2=-1.056674944
s3=-2.631705202
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

CDF – GAMMA

Returns the cumulative density of the Gamma distribution, based on the shape and scale parameters. This function is similar to PROBGAM where the optional argument in CDF – GAMMA is set to its default value.



Calculates the cumulative density function for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$k > 0, \theta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = \frac{\gamma(k, \frac{x}{\theta})}{\Gamma(k)}$$

$$\gamma(x, u) = \int_0^u t^{x-1} e^{-t} dt$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\gamma(x, u)$ is the lower incomplete Gamma function and $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = CDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = CDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = CDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = CDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=0.0136380006
s2=0.1558049836
s3=0.9599447964
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

PROBGAM

Returns the cumulative density of the Gamma distribution, based on the shape and scale parameters. This function is similar to CDF – GAMMA where the optional argument in CDF – GAMMA is set to its default value.

➤ **PROBGAM** ➤ (➤ **x** ➤ , ➤ **k** ➤) ➤ ➤

Calculates the cumulative density function of the Gamma distribution, based on the shape parameter k .

This function is defined under the following conditions:

$$x \geq 0, k > 0$$

$$\begin{cases} \text{if } x=0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; k) \end{cases}$$

$$f(x; k) = \frac{\gamma(k, x)}{\Gamma(k)}$$

$$\gamma(x, u) = \int_0^u t^{x-1} e^{-t} dt$$

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

where $\gamma(x, u)$ is the lower incomplete Gamma function and $\Gamma(x)$ is the Gamma function, see [GAMMA](#) [↗](#) (page 1813).

Note:

For negative values of x this function returns a missing value, whereas function CDF – GAMMA returns zero.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

Restriction: must be positive or zero

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PROBGAM (0.7,2);
  PUT s1=;
  s2 = PROBGAM (0.1,2);
  PUT s2=;
  s3 = PROBGAM (0.7,0.1);
  PUT s3=;
  s4 = PROBGAM (-3,2);
  PUT s4=;
  s5 = PROBGAM (0.7,-2);
  PUT s5=;
RUN;
```

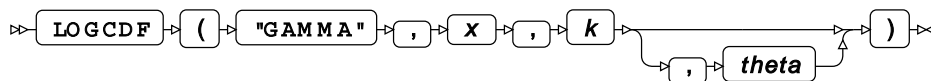
This produces the following output:

```
s1=0.1558049836
s2=0.0046788402
s3=0.9599447964
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGCDF – GAMMA

Returns the natural logarithm of the cumulative density of the Gamma distribution, based on the shape and scale parameters.



Calculates the natural logarithm of the cumulative density function for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$x > 0, k > 0, \theta > 0$$

$$f(x; k, \theta) = \log \frac{\gamma(k, \frac{x}{\theta})}{\Gamma(k)}$$

$$\gamma(x, u) = \int_0^u t^{x-1} e^{-t} dt$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\gamma(x, u)$ is the lower incomplete Gamma function and $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

k**Type:** Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1**Restriction:** must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = LOGCDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = LOGCDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = LOGCDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = LOGCDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

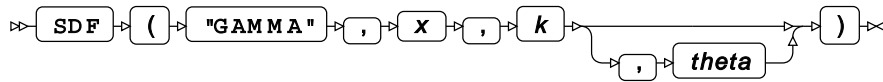
This produces the following output:

```
s1=-4.294895221
s2=-1.859150159
s3=-0.0408795
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

SDF – GAMMA

Returns the survival of the Gamma distribution, based on the shape and scale parameters.



Calculates the survival, or the complement to the cumulative density function, for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$k > 0, \theta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 1 \\ \text{otherwise} & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = 1 - \frac{\gamma(k, \frac{x}{\theta})}{\Gamma(k)}$$

$$\gamma(x, u) = \int_0^u t^{x-1} e^{-t} dt$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\gamma(x, u)$ is the lower incomplete Gamma function and $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = SDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = SDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = SDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = SDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

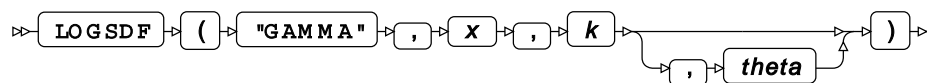
This produces the following output:

```
s1=0.9863619994
s2=0.8441950164
s3=0.0400552036
s4=1
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

LOGSDF – GAMMA

Returns the natural logarithm of the survival of the Gamma distribution, based on the shape and scale parameters.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Gamma distribution at point x , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$k > 0, \theta > 0$$

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; k, \theta) \end{cases}$$

$$f(x; k, \theta) = \log \left[1 - \frac{\gamma \left(k, \frac{x}{\theta} \right)}{\Gamma(k)} \right]$$

$$\gamma(x, u) = \int_0^u t^{x-1} e^{-t} dt$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

where $\gamma(x, u)$ is the lower incomplete Gamma function and $\Gamma(x)$ is the Gamma function, see [GAMMA](#) (page 1813).

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = LOGSDF ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = LOGSDF ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = LOGSDF ("GAMMA", -3, 2, 4);
  PUT s4=;
  s5 = LOGSDF ("GAMMA", 7, -1, 4);
  PUT s5=;
RUN;
```

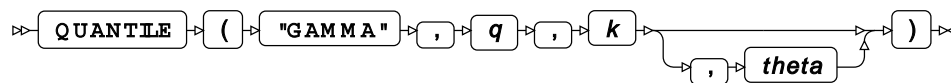
This produces the following output:

```
s1=-0.013731852
s2=-0.169371749
s3=-3.217496688
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output for negative values of x . The fifth example shows the output when x falls outside the domain bounds.

QUANTILE – GAMMA

Returns the quantile of the Gamma distribution, based on the shape and scale parameters. This function is similar to GAMINV where the optional argument in QUANTILE – GAMMA is set to its default value.



Calculates the quantile x , or the inverse of the cumulative density function, for the Gamma distribution for probability value q , based on the shape parameter k and scale parameter θ (*theta*). The scale parameter is optional; if it is omitted, it defaults to 1.

This function is defined under the following conditions:

$$0 < q < 1, k > 0, \theta > 0$$

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Gamma distribution, see section *CDF – GAMMA* [↗](#) (page 923).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE ("GAMMA", 0.7, 2, 4);
  PUT s1=;
  s2 = QUANTILE ("GAMMA", 0.7, 2, 1);
  PUT s2=;
  s3 = QUANTILE ("GAMMA", 0.7, 0.1, 1);
  PUT s3=;
  s4 = QUANTILE ("GAMMA", 0, 0.1, 1);
  PUT s4=;
  s5 = QUANTILE ("GAMMA", -3, 2, 4);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=9.7568659331
s2=2.4392164833
s3=0.0174277764
s4=.
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output when q equals to zero. The fifth example shows the output when an argument value falls outside the domain bounds.

GAMINV

Returns the quantile of the Gamma distribution, based on the shape and scale parameters. This function is similar to QUANTILE – GAMMA where the optional argument in QUANTILE – GAMMA is set to its default value.

➤ **GAM INV** ➤ (➤ q ➤ , ➤ k ➤) ➤

Calculates the quantile x , or the inverse of the cumulative density function, for the Gamma distribution for probability value q , based on the shape parameter k .

This function is defined under the following conditions:

$$0 \leq q < 1, k > 0$$

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Gamma distribution, see section *CDF – GAMMA* [↗](#) (page 923).

Note:

In this function, the probability q can equal 0 (zero), whereas in function QUANTILE – GAMMA q must be strictly greater than 0 (zero).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

k

Type: Numeric

The shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Gamma distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = GAMINV (0.7,2);
  PUT s1=;
  s2 = GAMINV (0.1,2);
  PUT s2=;
  s3 = GAMINV (0.7,0.1);
  PUT s3=;
  s4 = GAMINV (0,0.1);
  PUT s4=;
  s5 = GAMINV (-3,2);
  PUT s5=;
RUN;
```

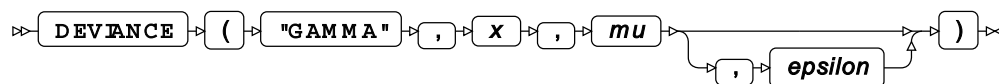
This produces the following output:

```
s1=2.4392164833
s2=0.5318116084
s3=0.0174277764
s4=0
s5=.
```

The first three examples show the effects of the arguments on the result. The fourth example shows the output when q equals to zero. The fifth example shows the output when an argument value falls outside the domain bounds.

DEVIANC – GAMMA

Returns the deviance of the Gamma distribution at a specified point, based on the distribution mean.



Calculates the deviance, or goodness of fit, for the generalised linear model of the Gamma distribution at a nonnegative point x based on the distribution mean μ (mu). An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The distribution mean is then adjusted so that $\mu \geq \varepsilon$:

$$\text{if } \mu < \varepsilon \text{ set } \mu = \varepsilon$$

If $x \geq 0$, it is adjusted so that $x \geq \varepsilon$:

$$\text{if } 0 \leq x < \varepsilon \text{ set } x = \varepsilon$$

These adjusted values of μ and x are used in the subsequent calculation of the deviance.

$$2\left(\frac{x-\mu}{\mu} - \log \frac{x}{\mu}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

mu

Type: Numeric

The distribution mean.

Expected: $\mu > 0$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$

Expected: $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Examples – applying correction to the distribution mean

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("GAMMA", 0.1, 0.0007, 0.0005);  
  PUT g1=;  
  g2 = DEVIANCE("GAMMA", 0.1, 0.0007, 0.0010);  
  PUT g2=;  
  g3 = DEVIANCE("GAMMA", 0.1, 0.0007, 0.0015);  
  PUT g3=;  
  g4 = DEVIANCE("GAMMA", 0.1, 0.0007          );  
  PUT g4=;  
RUN;
```

This produces the following output:

```
g1=273.79059545  
g2=188.78965963  
g3=122.93392318  
g4=273.79059545
```

The value of the distribution mean is not corrected in the first example because $\mu > \varepsilon$. However, this condition does not hold in the second and third example, and correction is applied: $\mu = \varepsilon$. This corrected value is used for calculation, yielding different results.

In the fourth example the ε parameter is omitted, so the default value of $\varepsilon = 10^{-12}$ is used. Here, as in the first example, $\mu > \varepsilon$, so no correction is required.

Examples – invariant scaling

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("GAMMA", 0.001, 0.007);  
  PUT g1=;  
  g2 = DEVIANCE("GAMMA", 0.01 , 0.07 );  
  PUT g2=;  
  g3 = DEVIANCE("GAMMA", 0.1   , 0.7   );  
  PUT g3=;  
RUN;
```

This produces the following output:

```
g1=6.5255619127  
g2=6.5255619127  
g3=6.5255619127
```

For the Gamma distribution, when both the point of measurement x and the distribution mean μ are scaled with the same factor, the deviance remains the same.

RAND – GAMMA

Returns a random number from the Gamma distribution based on the shape. This function is similar to RANGAM and CALL RANGAM.



The distribution is parameterised using a scale of 1.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

shape

Type: Numeric

A shape parameter for the distribution.

Example

In this example, a random number from the Gamma distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("GAMMA", 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
5.7003495538
2.9098612135
0.7765584912
1.9572029415
1.0605257842
```


Running the `DATA` step again produces the following output.

```
The random numbers are:
1.95537115
0.2755466634
2.9501199923
4.4732083311
2.9515196911
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(12);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("GAMMA", 3);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
3.4869816111
2.1891910706
2.8123690729
3.0755814295
2.3247368117
```

Running the `DATA` step again produces the same output.

RANGAM

Returns a random number from the Gamma distribution based on the shape. This function is similar to `RAND – GAMMA` and `CALL RANGAM`.

➤ **RANGAM** ➤ (➤ *seed* ➤ , ➤ *shape* ➤) ➤ ➤

The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is positive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

shape

Type: Numeric

A positive number that defines the shape of the gamma distribution.

If a negative number is used, a missing value is returned from the function, and a note indicating the argument is invalid is written to the log.

Example

In this example, a random number from the Gamma distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANGAM(50, 0.2);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.0104523162  
0.5938802889  
0.0000965085  
0.0008535642  
0.621719791
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANGAM(0, 0.2);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
6.728089E-10  
0.0538443586  
0.0131700204  
0.6767632093  
0.0109073109
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.0002528525  
1.1553658321  
0.0003353805  
0.1849489086  
2.9069978738
```

CALL RANGAM

Returns a random number from the Gamma distribution based on the shape. This routine is similar to function RAND – GAMMA and RANGAM.

```
CALL RANGAM ( seed , shape , x ) ;
```

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a DATA step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the DATA step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same DATA step. To generate a different sequence of random numbers each time the DATA step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is positive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

shape**Type:** Numeric

A value defining the shape of the gamma distribution.

x**Type:** Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Gamma distribution based on the shape is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    call rangam(50, 0.2, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.0104523162  
0.5938802889  
0.0000965085  
0.0008535642  
0.621719791
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    call rangam(0, 0.2, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.0577740856  
1.073869E-7  
0.3667352159  
0.0350047029  
2.4169802E-6
```

Running the DATA step again produces the following output.

```
The random numbers are:
3.1296626E-6
0.001638255
0.4199521379
0.0057071644
0.0926429502
```

Gaussian distribution

Functions for the Gaussian distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – GAUSSIAN [↗](#)..... 945

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Returns the value of the probability density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of PMF – GAUSSIAN, PDF – NORMAL and PMF – NORMAL.

PMF – GAUSSIAN [↗](#)..... 947

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Returns the value of the probability mass function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of PDF – GAUSSIAN, PDF – NORMAL and PMF – NORMAL.

LOGPDF – GAUSSIAN [↗](#)..... 949

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

Returns the value of the natural logarithm of the probability density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGPMF – GAUSSIAN, LOGPDF – NORMAL and LOGPMF – NORMAL.

LOGPMF – GAUSSIAN [↗](#)..... 951

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

Returns the value of the natural logarithm of the probability mass function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – GAUSSIAN, LOGPDF – NORMAL and LOGPMF – NORMAL.

CDF – GAUSSIAN [↗](#)..... 953

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

Returns the value of the cumulative density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of CDF – NORMAL and is similar to PROBNORM.

LOGCDF – GAUSSIAN [↗](#)..... 955

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt\right)$$

Returns the value of the natural logarithm of the cumulative density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGCDF – NORMAL.

SDF – GAUSSIAN [↗](#)..... 957

$$1 - \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

Returns the value of the survival function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of SDF – NORMAL.

LOGSDF – GAUSSIAN [↗](#)..... 960

$$\log\left(1 - \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt\right)$$

Returns the value of the natural logarithm of the survival function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGSDF – NORMAL.

QUANTILE – GAUSSIAN [↗](#)..... 962

$$\inf \{x: q \leq \text{CDF}(x; \mu, \sigma)\}$$

Returns the value of the quantile function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of QUANTILE – NORMAL and is similar to PROBIT.

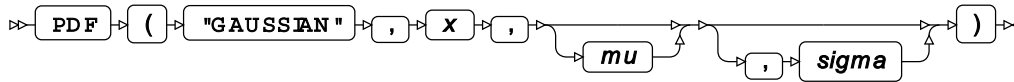
DEVIANCE – GAUSSIAN [↗](#)..... 964

$$(x - \mu)^2$$

Returns the deviance of the Gaussian distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – NORMAL.

PDF – GAUSSIAN

Returns the value of the probability density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of PMF – GAUSSIAN, PDF – NORMAL and PMF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the probability density function for the standard Gaussian distribution at point x .

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the probability density function of the Gaussian distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = PDF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = PDF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = PDF("GAUSSIAN", 0);
  PUT s3=;
  s4 = PDF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = PDF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = PDF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = PDF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = PDF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=0.3989422804
s2=0.3989422804
s3=0.3989422804
s4=0.0647587978
s5=0.0647587978
s6=.
s7=.
s8=.
```

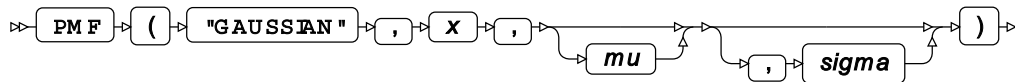
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the probability density function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the probability density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

PMF – GAUSSIAN

Returns the value of the probability mass function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of PDF – GAUSSIAN, PDF – NORMAL and PMF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the probability mass function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the probability mass function of the Gaussian distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = PMF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = PMF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = PMF("GAUSSIAN", 0);
  PUT s3=;
  s4 = PMF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = PMF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = PMF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = PMF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = PMF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=0.3989422804
s2=0.3989422804
s3=0.3989422804
s4=0.0647587978
s5=0.0647587978
s6=.
s7=.
s8=.
```

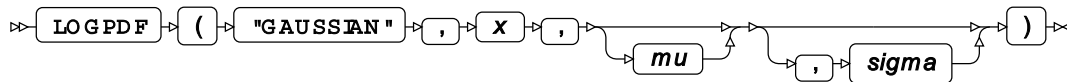
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the probability mass function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the probability mass function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

LOGPDF – GAUSSIAN

Returns the value of the natural logarithm of the probability density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGPMF – GAUSSIAN, LOGPDF – NORMAL and LOGPMF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the probability density function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the probability density function of the Gaussian distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = LOGPDF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = LOGPDF("GAUSSIAN", 0);
  PUT s3=;
  s4 = LOGPDF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = LOGPDF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = LOGPDF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGPDF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = LOGPDF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.918938533
s2=-0.918938533
s3=-0.918938533
s4=-2.737085714
s5=-2.737085714
s6=.
s7=.
s8=.
```

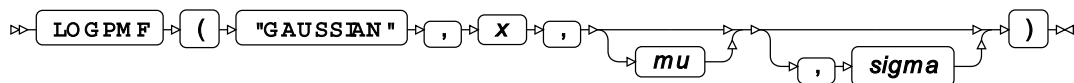
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the probability density function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the probability density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

LOGPMF – GAUSSIAN

Returns the value of the natural logarithm of the probability mass function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – GAUSSIAN, LOGPDF – NORMAL and LOGPMF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the probability mass function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log \left(\frac{1}{\sigma \sqrt{2\pi}} \exp \left(-\frac{1}{2} \left(\frac{x - \mu}{\sigma} \right)^2 \right) \right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log \left(\frac{1}{\sqrt{2\pi}} \exp \left(-\frac{1}{2} x^2 \right) \right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the probability mass function of the Gaussian distribution is calculated for various values of *x*, *mu* and *sigma*. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = LOGPMF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = LOGPMF("GAUSSIAN", 0);
  PUT s3=;
  s4 = LOGPMF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = LOGPMF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = LOGPMF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGPMF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = LOGPMF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.918938533
s2=-0.918938533
s3=-0.918938533
s4=-2.737085714
s5=-2.737085714
s6=.
s7=.
s8=.
```

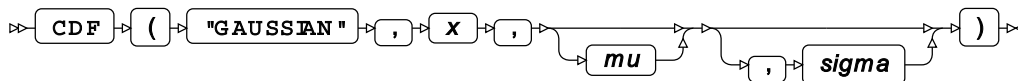
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the probability mass function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the probability mass function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

CDF – GAUSSIAN

Returns the value of the cumulative density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of CDF – NORMAL and is similar to PROB NORM.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the cumulative density function for the standard Gaussian distribution.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x

Type: Numeric

The point at which to calculate the cumulative density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the cumulative density function of the Gaussian distribution is calculated for various values of *x*, *mu* and *sigma*. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = CDF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = CDF("GAUSSIAN", 0);
  PUT s3=;
  s4 = CDF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = CDF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = CDF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = CDF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = CDF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```


This produces the following output:

```
s1=0.5
s2=0.5
s3=0.5
s4=0.0668072013
s5=0.0668072013
s6=.
s7=.
s8=.
```

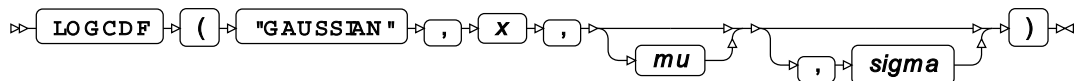
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the cumulative density function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the cumulative density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

LOGCDF – GAUSSIAN

Returns the value of the natural logarithm of the cumulative density function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGCDF – NORMAL.



You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the cumulative density function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log \left(\frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} \left(\frac{t - \mu}{\sigma} \right)^2 \right) dt \right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log \left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} t^2 \right) dt \right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the cumulative density function of the Gaussian distribution is calculated for various values of *x*, *mu* and *sigma*. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = LOGCDF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = LOGCDF("GAUSSIAN", 0);
  PUT s3=;
  s4 = LOGCDF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = LOGCDF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = LOGCDF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGCDF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = LOGCDF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.693147181
s2=-0.693147181
s3=-0.693147181
s4=-2.705944401
s5=-2.705944401
s6=.
s7=.
s8=.
```

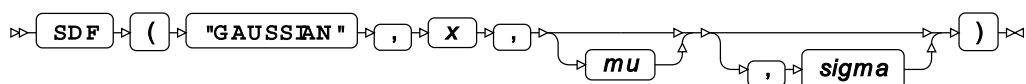
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the cumulative density function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the cumulative density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

SDF – GAUSSIAN

Returns the value of the survival function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of SDF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the survival function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = 1 - \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x

Type: Numeric

The point at which to calculate the value of the survival function.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the survival function of the Gaussian distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;  
  s1 = SDF("GAUSSIAN", 0, 0, 1);  
  PUT s1=;  
  s2 = SDF("GAUSSIAN", 0, 0);  
  PUT s2=;  
  s3 = SDF("GAUSSIAN", 0);  
  PUT s3=;  
  s4 = SDF("GAUSSIAN", -3, 0, 2);  
  PUT s4=;  
  s5 = SDF("GAUSSIAN", 7, 10, 2);  
  PUT s5=;  
  s6 = SDF("GAUSSIAN", -3, 0, 2, 3);  
  PUT s6=;  
  s7 = SDF("GAUSSIAN", -3, , 2);  
  PUT s7=;  
  s8 = SDF("GAUSSIAN", -3, 0, 0);  
  PUT s8=;  
RUN;
```

This produces the following output:

```
s1=0.5  
s2=0.5  
s3=0.5  
s4=0.9331927987  
s5=0.9331927987  
s6=.  
s7=.  
s8=.
```

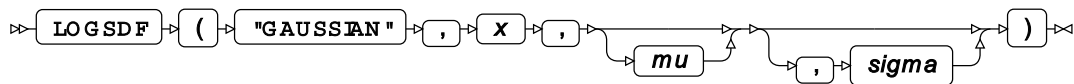
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the survival function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the survival function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

LOGSDF – GAUSSIAN

Returns the value of the natural logarithm of the survival function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of LOGSDF – NORMAL.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the survival function for the standard Gaussian distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Gaussian distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log \left(1 - \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} \left(\frac{t - \mu}{\sigma} \right)^2 \right) dt \right)$$

and the calculated value for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log \left(1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} t^2 \right) dt \right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the distribution.

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the survival function of the Gaussian distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF("GAUSSIAN", 0, 0, 1);
  PUT s1=;
  s2 = LOGSDF("GAUSSIAN", 0, 0);
  PUT s2=;
  s3 = LOGSDF("GAUSSIAN", 0);
  PUT s3=;
  s4 = LOGSDF("GAUSSIAN", -3, 0, 2);
  PUT s4=;
  s5 = LOGSDF("GAUSSIAN", 7, 10, 2);
  PUT s5=;
  s6 = LOGSDF("GAUSSIAN", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGSDF("GAUSSIAN", -3, , 2);
  PUT s7=;
  s8 = LOGSDF("GAUSSIAN", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.693147181
s2=-0.693147181
s3=-0.693147181
s4=-0.069143456
s5=-0.069143456
s6=.
s7=.
s8=.
```

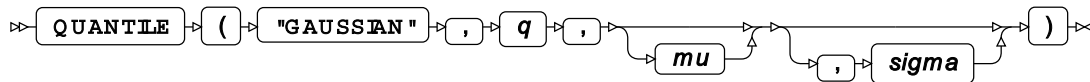
The first three examples all specify the same point, 0, in a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the survival function.

The fourth example specifies a point, -3, in a Gaussian distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Gaussian distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the survival function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

QUANTILE – GAUSSIAN

Returns the value of the quantile function at a given point for the Gaussian distribution with the specified mean and standard deviation. This function is an alias of QUANTILE – NORMAL and is similar to PROBIT.



The Gaussian distribution is also known as the Normal distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the quantile function for the standard Gaussian distribution at *q*.

This function is defined for

$$\begin{cases} 0 < q < 1 \\ \sigma > 0 \end{cases}$$

The calculated value, x , for the Gaussian distribution with mean μ and standard deviation σ , is:

$$f(q; \mu, \sigma) = \inf \{x: q \leq \text{CDF}(x; \mu, \sigma)\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ (*infinium*) is the greatest lower bound of x , and $\text{CDF}(x, \mu, \sigma)$ is the cumulative density function of the Gaussian distribution with mean μ and standard deviation σ .

The calculated value, x , for the standard Gaussian distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ is the greatest lower bound of x , and $\text{CDF}(x)$ is the cumulative density function of the standard Gaussian distribution.

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the distribution.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the quantile function of the Gaussian distribution is calculated for various values of *q*, *mu* and *sigma*. The results are written to the log.

As this function is the inverse of the cumulative density function, this example demonstrates the relationship by using the same points in the distribution as the cumulative density function example, *CDF – GAUSSIAN* [↗](#) (page 953). Therefore the *q* values in this example are the returned values in the cumulative density function example, and the returned values in this example are the *x* values in the cumulative density function example (subject to rounding errors).

```
DATA _NULL_;
  s1 = QUANTILE("GAUSSIAN", 0.5, 0, 1);
  PUT s1=;
  s2 = QUANTILE("GAUSSIAN", 0.5, 0);
  PUT s2=;
  s3 = QUANTILE("GAUSSIAN", 0.5);
  PUT s3=;
  s4 = QUANTILE("GAUSSIAN", 0.0668072013, 0, 2);
  PUT s4=;
  s5 = QUANTILE("GAUSSIAN", 0.0668072013, 10, 2);
  PUT s5=;
  s6 = QUANTILE("GAUSSIAN", 0.0668072013, 0, 2, 3);
  PUT s6=;
  s7 = QUANTILE("GAUSSIAN", 0.0668072013, , 2);
  PUT s7=;
  s8 = QUANTILE("GAUSSIAN", 0.0668072013, 0, 0);
  PUT s8=;
  s9 = QUANTILE("GAUSSIAN", 1.0, 0, 1);
  PUT s9=;
RUN;
```

This produces the following output:

```
s1=0
s2=0
s3=0
s4=-3
s5=7.0000000005
s6=.
s7=.
s8=.
s9=.
```

The first three examples all specify the same cumulative density value, 0.5, for a standard Gaussian distribution with mean 0 and standard deviation 1. So they all return the same value for the quantile function. In this case, the value is the point in the distribution where the probability is 0.5 that a random member of the distribution falls below this point. For the standard Gaussian distribution, this value is the mean, 0.

The fourth example specifies a cumulative density value of 0.0668072013, a mean of 0 and a standard deviation of 2. The fifth example specifies the same cumulative density value and standard deviation, but has a mean of 10. For a Gaussian distribution with a standard deviation of 2, 0.0668072013 is the value of the cumulative density function at a point 1.5 standard deviations below the mean. So both these examples return the point that is 1.5 standard deviations below the mean, although the actual values returned are different because the examples have different means. There is also a small rounding error in the last digit of the fifth example.

The last four examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, the eighth example specifies an invalid value for *sigma* and the ninth example specifies an invalid value for *q*, the cumulative density.

DEVIANCE – GAUSSIAN

Returns the deviance of the Gaussian distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – NORMAL.

⇒ **DEVIANCE** ⇒ ("GAUSSIAN" , *x* , *mu*) ⇒

Keyword **GAUSSIAN** is an alias of **NORMAL**, see **DEVIANCE – NORMAL** [↗](#) (page 1202).

Calculates the deviance, or goodness of fit, for the generalised linear model of the Gaussian distribution at point *x* based on the distribution mean μ (*mu*).

$$(x - \mu)^2$$

Return type: Numeric

If the wrong number of arguments are supplied, a missing value is returned.

x**Type:** Numeric

The point at which to calculate the deviance.

If the argument contains a missing value, a missing value is returned.

mu**Type:** Numeric

The distribution mean.

If the argument contains a missing value, a missing value is returned.

Basic examples

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = DEVIANCE("GAUSSIAN", -12.3, 0);  
  PUT s1=;  
  s2 = DEVIANCE("GAUSSIAN", 1.1, 1.1);  
  PUT s2=;  
RUN;
```

This produces the following output:

```
s1=151.29  
s2=0
```

The second example demonstrates that when the point of measurement equals the distribution mean, the deviance of the Gaussian distribution is zero.

Examples – proportional scaling

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("GAUSSIAN", 0.01, 0.07);  
  PUT g1=;  
  g2 = DEVIANCE("GAUSSIAN", 0.1 , 0.7 );  
  PUT g2=;  
  g3 = DEVIANCE("GAUSSIAN", 1 , 7 );  
  PUT g3=;  
RUN;
```

This produces the following output:

```
g1=0.0036  
g2=0.36  
g3=36
```

Due to the nature of the Gaussian distribution, when both the point of measurement x and the distribution mean μ are scaled with the same factor, the deviance is scaled with the square of that factor.

Geometric distribution

Functions for the Geometric distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – GEOMETRIC [↗](#)..... 967
 $(1-p)^r p$

Returns the probability density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of PMF – GEOMETRIC.

PMF – GEOMETRIC [↗](#)..... 968
 $(1-p)^r p$

Returns the probability mass of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of PDF – GEOMETRIC.

LOGPDF – GEOMETRIC [↗](#)..... 970
 $\log[(1-p)^r p]$

Returns the natural logarithm of the probability density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of LOGPMF – GEOMETRIC.

LOGPMF – GEOMETRIC [↗](#)..... 971
 $\log[(1-p)^r p]$

Returns the natural logarithm of the probability mass of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of LOGPDF – GEOMETRIC.

CDF – GEOMETRIC [↗](#)..... 973
 $1 - (1-p)^{r+1}$

Returns the cumulative density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

LOGCDF – GEOMETRIC [↗](#)..... 974
 $\log[1 - (1-p)^{r+1}]$

Returns the natural logarithm of the cumulative density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

SDF – GEOMETRIC ↗	976
$(1-p)^{r+1}$	
Returns the survival of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.	
LOGSDF – GEOMETRIC ↗	977
$\log[(1-p)^{r+1}](1-p)^{r+1}$	
Returns the natural logarithm of the survival of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.	
QUANTILE – GEOMETRIC ↗	979
$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$	
Returns the quantile of the Geometric distribution for a specified probability value, based on the probability of success in Bernoulli trials.	
RAND – GEOMETRIC ↗	980
Returns a random number from the Geometric distribution based on the probability.	

PDF – GEOMETRIC

Returns the probability density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of PMF – GEOMETRIC.

➤ PDF ➤ (➤ "GEOMETRIC" ➤ , ➤ *r* ➤ , ➤ *p* ➤) ➤ ➤

Calculates the probability density function for the Geometric distribution for the number of failures *r* based on the probability of success *p* in Bernoulli trials.

$$\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p; r) \end{cases}$$

$$f(p; r) = (1-p)^r p$$

Return type: Numeric

The return value is between 0 and 1, inclusive.

r

Type: Numeric

The number of failures until the first success.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("GEOMETRIC", 3, 0.7);
  PUT s1=;
  s2 = PDF ("GEOMETRIC", -2, 0.7);
  PUT s2=;
  s3 = PDF ("GEOMETRIC", 3, 0);
  PUT s3=;
  s4 = PDF ("GEOMETRIC", 3, 1);
  PUT s4=;
  s5 = PDF ("GEOMETRIC", 0.6, 0.7);
  PUT s5=;
  s6 = PDF ("GEOMETRIC", 3, 2);
  PUT s6=;
RUN;
```

This produces the following output:

```
s1=0.0189
s2=0
s3=0
s4=0
s5=.
s6=.
```

The first four examples show the output when the arguments lie within the domain bounds. The last two examples show the output when one of the arguments falls outside the domain bounds.

PMF – GEOMETRIC

Returns the probability mass of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of PDF – GEOMETRIC.

➤ **PMF** ➤ (➤ **"GEOMETRIC"** ➤ , ➤ *r* ➤ , ➤ *p* ➤) ➤ ➤

Calculates the probability mass function for the Geometric distribution for the number of failures *r* based on the probability of success *p* in Bernoulli trials.

$$\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p; r) \end{cases}$$

$$f(p; r) = (1-p)^r p$$

Return type: Numeric

The return value is between 0 and 1, inclusive.

r

Type: Numeric

The number of failures until the first success.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  
s1 = PMF ("GEOMETRIC", 3, 0.7);  
PUT s1=;  
s2 = PMF ("GEOMETRIC", -2, 0.7);  
PUT s2=;  
s3 = PMF ("GEOMETRIC", 3, 0);  
PUT s3=;  
s4 = PMF ("GEOMETRIC", 3, 1);  
PUT s4=;  
s5 = PMF ("GEOMETRIC", 0.6, 0.7);  
PUT s5=;  
s6 = PMF ("GEOMETRIC", 3, 2);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s1=0.0189  
s2=0  
s3=0  
s4=0  
s5=.  
s6=.
```

The first four examples show the output when the arguments lie within the domain bounds. The last two examples show the output when one of the arguments falls outside the domain bounds.

LOGPDF – GEOMETRIC

Returns the natural logarithm of the probability density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of LOGPMF – GEOMETRIC.

» LOGPDF ("GEOMETRIC" , *r* , *p*) «

Calculates the natural logarithm of the probability density function for the Geometric distribution for the number of failures *r* based on the probability of success *p* in Bernoulli trials.

$$f(p; r) = \log[(1-p)^r p] = r \log(1-p) + \log p$$

Return type: Numeric

The return value is negative or zero.

r

Type: Numeric

The number of failures until the first success.

Restriction: $r \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 < p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF ("GEOMETRIC", 3, 0.7);
PUT s1=;
s2 = LOGPDF ("GEOMETRIC", -2, 0.7);
PUT s2=;
s3 = LOGPDF ("GEOMETRIC", 3, 0);
PUT s3=;
s4 = LOGPDF ("GEOMETRIC", 3, 1);
PUT s4=;
s5 = LOGPDF ("GEOMETRIC", 0.6, 0.7);
PUT s5=;
s6 = LOGPDF ("GEOMETRIC", 3, 2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=-3.968593357
s2=.
s3=.
s4=.
s5=.
s6=.
```

The first example shows the output when the arguments lie within the domain bounds. The remaining examples show the output when one of the arguments falls outside the domain bounds. The argument values in these examples are the same as in the examples for function PDF – GEOMETRIC.

LOGPMF – GEOMETRIC

Returns the natural logarithm of the probability mass of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials. This function is an alias of LOGPDF – GEOMETRIC.

```
LOGPMF ( "GEOMETRIC" , r , p )
```

Calculates the natural logarithm of the probability mass function for the Geometric distribution for the number of failures r based on the probability of success p in Bernoulli trials.

$$f(p; r) = \log [(1-p)^r p] = r \log (1-p) + \log p$$

Return type: Numeric

The return value is negative or zero.

r**Type:** Numeric

The number of failures until the first success.

Restriction: $r \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

p**Type:** Numeric

The probability of success for each trial.

Restriction: $0 < p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPMF ("GEOMETRIC", 3, 0.7);
PUT s1=;
s2 = LOGPMF ("GEOMETRIC", -2, 0.7);
PUT s2=;
s3 = LOGPMF ("GEOMETRIC", 3, 0);
PUT s3=;
s4 = LOGPMF ("GEOMETRIC", 3, 1);
PUT s4=;
s5 = LOGPMF ("GEOMETRIC", 0.6, 0.7);
PUT s5=;
s6 = LOGPMF ("GEOMETRIC", 3, 2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=-3.968593357
s2=.
s3=.
s4=.
s5=.
s6=.
```

The first example shows the output when the arguments lie within the domain bounds. The remaining examples show the output when one of the arguments falls outside the domain bounds. The argument values in these examples are the same as in the examples for function PMF – GEOMETRIC.

CDF – GEOMETRIC

Returns the cumulative density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

⇒ CDF ⇒ (⇒ "GEOMETRIC" ⇒ , ⇒ *r* ⇒ , ⇒ *p* ⇒) ⇒

Calculates the cumulative density function for the Geometric distribution for the number of failures *r* based on the probability of success *p* in Bernoulli trials.

$$f(p; r) = 1 - (1 - p)^{r+1}$$

Return type: Numeric

The return value is less than or equal to one.

r

Type: Numeric

The number of failures until the first success.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

  s1 = CDF ("GEOMETRIC",3,0.7);
  PUT s1=;
  s2 = CDF ("GEOMETRIC",-2,0.7);
  PUT s2=;
  s3 = CDF ("GEOMETRIC",3,0);
  PUT s3=;
  s4 = CDF ("GEOMETRIC",3,1);
  PUT s4=;
  s5 = CDF ("GEOMETRIC",0.6,0.7);
  PUT s5=;
  s6 = CDF ("GEOMETRIC",3,2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.9919
s2=-2.333333333
s3=0
s4=1
s5=.
s6=.
```

The first four examples show the output when the arguments lie within the domain bounds. The last two examples show the output when one of the arguments falls outside the domain bounds.

LOGCDF – GEOMETRIC

Returns the natural logarithm of the cumulative density of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

```
LOGCDF ( "GEOMETRIC" , r , p )
```

Calculates the natural logarithm of the cumulative density function for the Geometric distribution for the number of failures r based on the probability of success p in Bernoulli trials.

$$\begin{cases} \text{if } p=1 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p;r) \end{cases}$$

$$f(p;r) = \log[1 - (1-p)^{r+1}]$$

Return type: Numeric

The return value is negative or zero.

r**Type:** Numeric

The number of failures until the first success.

Restriction: $r \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

p**Type:** Numeric

The probability of success for each trial.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF ("GEOMETRIC", 3, 0.7);
PUT s1=;
s2 = LOGCDF ("GEOMETRIC", -2, 0.7);
PUT s2=;
s3 = LOGCDF ("GEOMETRIC", 3, 0);
PUT s3=;
s4 = LOGCDF ("GEOMETRIC", 3, 1);
PUT s4=;
s5 = LOGCDF ("GEOMETRIC", 0.6, 0.7);
PUT s5=;
s6 = LOGCDF ("GEOMETRIC", 3, 2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=-0.008132983
s2=.
s3=.
s4=0
s5=.
s6=.
```

The first and fourth examples show the output when the arguments lie within the domain bounds. The remaining examples show the output when one of the arguments falls outside the domain bounds. The argument values in these examples are the same as in the examples for function CDF – GEOMETRIC.

SDF – GEOMETRIC

Returns the survival of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

Diagram illustrating the function syntax: `SDF ("GEOMETRIC" , r , p)`

Calculates the survival, or the complement to the cumulative density function, for the Geometric distribution for the number of failures r based on the probability of success p in Bernoulli trials.

$$f(p; r) = (1-p)^{r+1}$$

Return type: Numeric

The return value is positive or zero.

r

Type: Numeric

The number of failures until the first success.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

  s1 = SDF ("GEOMETRIC",3,0.7);
  PUT s1=;
  s2 = SDF ("GEOMETRIC",-2,0.7);
  PUT s2=;
  s3 = SDF ("GEOMETRIC",3,0);
  PUT s3=;
  s4 = SDF ("GEOMETRIC",3,1);
  PUT s4=;
  s5 = SDF ("GEOMETRIC",0.6,0.7);
  PUT s5=;
  s6 = SDF ("GEOMETRIC",3,2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.0081
s2=3.333333333
s3=1
s4=0
s5=.
s6=.
```

The first four examples show the output when the arguments lie within the domain bounds. The last two examples show the output when one of the arguments falls outside the domain bounds.

LOGSDF – GEOMETRIC

Returns the natural logarithm of the survival of the Geometric distribution for a specified number of failures, based on the probability of success in Bernoulli trials.

```
LOGSDF ( "GEOMETRIC" , r , p )
```

Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Geometric distribution for the number of failures r based on the probability of success p in Bernoulli trials.

$$f(p; r) = \begin{cases} \text{return } 0 & \text{if } p=0 \\ \text{return } f(p; r) & \text{otherwise} \end{cases}$$

$$f(p; r) = \log [(1-p)^{r+1}] = (r+1) \log (1-p)$$

Return type: Numeric

r**Type:** Numeric

The number of failures until the first success.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

p**Type:** Numeric

The probability of success for each trial.

Restriction: $0 < p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGSDF ("GEOMETRIC", 3, 0.7);
  PUT s1=;
  s2 = LOGSDF ("GEOMETRIC", -2, 0.7);
  PUT s2=;
  s3 = LOGSDF ("GEOMETRIC", 3, 0);
  PUT s3=;
  s4 = LOGSDF ("GEOMETRIC", 3, 1);
  PUT s4=;
  s5 = LOGSDF ("GEOMETRIC", 0.6, 0.7);
  PUT s5=;
  s6 = LOGSDF ("GEOMETRIC", 3, 2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=-4.815891217
s2=1.2039728043
s3=0
s4=.
s5=.
s6=.
```

The first three examples show the output when the arguments lie within the domain bounds. The remaining examples show the output when one of the arguments falls outside the domain bounds. The argument values in these examples are the same as in the examples for function SDF – GEOMETRIC.

QUANTILE – GEOMETRIC

Returns the quantile of the Geometric distribution for a specified probability value, based on the probability of success in Bernoulli trials.

⇒ QUANTILE ("GEOMETRIC" , q , p) ⇒

Calculates the quantile x , or the inverse of the cumulative density function, for the Geometric distribution for probability value q based on the probability of success p in Bernoulli trials.

$$\begin{cases} \text{if } q=0 & \text{return } 0 \\ \text{if } p=1 & \text{return } 0 \\ \text{otherwise} & \text{return } f(q) \end{cases}$$

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Geometric distribution, see section *CDF – GEOMETRIC* [↗](#) (page 973).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

p

Type: Numeric

The probability of success for each trial.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Geometric distribution is returned. The results are written to the log.

```
DATA _NULL_;

  s1 = QUANTILE ("GEOMETRIC", 0.973, 0.7);
  PUT s1=;
  s2 = QUANTILE ("GEOMETRIC", 0.974, 0.7);
  PUT s2=;
  s3 = QUANTILE ("GEOMETRIC", 0.6, 0.7);
  PUT s3=;
  s4 = QUANTILE ("GEOMETRIC", -3, 0.7);
  PUT s4=;

RUN;
```

This produces the following output:

```
s1=2
s2=3
s3=0
s4=.
```

The first three examples show the output when the arguments lie within the domain bounds. The last example show the output when one of the arguments falls outside the domain bounds.

RAND – GEOMETRIC

Returns a random number from the Geometric distribution based on the probability.

```
>> RAND ( "GEOMETRIC" , probability ) <<
```

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

probability

Type: Numeric

A probability from the distribution.

Example

In this example, a random number from the Geometric distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("GEOMETRIC", 0.75);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
1.0408202604  
0.6931654616  
2.1977668027  
0.3678618118  
0.3735144205
```

Running the DATA step again produces the following output.

```
The random numbers are:  
1.6421995056  
1.9395878658  
0.7045833721  
0.0342775477  
0.2960285254
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(11);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("GEOMETRIC", 0.75);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
2.0659785187  
0.5051382886  
1.0388574894  
0.9438183502  
0.3738827807
```

Running the DATA step again produces the same output.

Gumbel distribution

Functions for the Gumbel distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – GUMBEL [↗](#)..... 983

$$\frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of PMF – GUMBEL.

PMF – GUMBEL [↗](#)..... 985

$$\frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of PDF – GUMBEL.

LOGPDF – GUMBEL [↗](#)..... 988

$$-\log\beta - (z + e^{-z}) \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the natural logarithm of the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of LOGPMF – GUMBEL.

LOGPMF – GUMBEL [↗](#)..... 990

$$-\log\beta - (z + e^{-z}) \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the natural logarithm of the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of LOGPDF – GUMBEL.

CDF – GUMBEL [↗](#)..... 993

$$\exp(-e^{-z}) \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the cumulative density at a point from a Gumbel distribution, with specified location and scale parameters.

LOGCDF – GUMBEL [↗](#)..... 995

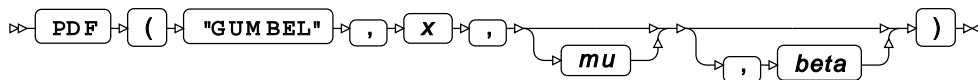
$$-e^{-z} \text{ where } z = \frac{x - \mu}{\beta}$$

Returns the natural logarithm of the cumulative density at a point from a Gumbel distribution, with specified location and scale parameters.

SDF – GUMBEL ↗	997
$1 - \exp(-e^{-z})$ where $z = \frac{x - \mu}{\beta}$	
Returns the survival density at a point from a Gumbel distribution, with specified location and scale parameters.	
LOGSDF – GUMBEL ↗	1000
$\log(1 - \exp(-e^{-z}))$ where $z = \frac{x - \mu}{\beta}$	
Returns the natural logarithm of the survival density at a point from a Gumbel distribution, with specified location and scale parameters.	
QUANTILE – GUMBEL ↗	1002
$\inf \{x : q \leq \text{CDF}(x, \mu, \beta)\}$	
Returns the quantile of a Gumbel distribution with specified location and scale parameters, for a specified cumulative density.	
RAND – GUMBEL ↗	1004
Returns a random number from the Gumbel distribution.	

PDF – GUMBEL

Returns the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of PMF – GUMBEL.



The probability density function at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The probability density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: *beta* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= PDF ("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= PDF ("GUMBEL", 1, 0);  
PUT s2=;  
s3= PDF ("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=0.25464638  
s2=0.25464638  
s3=0.25464638
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = PDF ("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.0605123465
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a much lower probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
s5 = PDF("GUMBEL", 5, 0, 1);
PUT s5=;
s6 = PDF("GUMBEL", 2, -3, 1);
PUT s6=;
RUN;
```

This produces the following output:

```
s5=0.0066926997
s6=0.0066926997
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = PDF("GUMBEL", 1, 1, 0, 0);
PUT s7=;
s8 = PDF("GUMBEL", 1, , 0);
PUT s8=;
RUN;
```

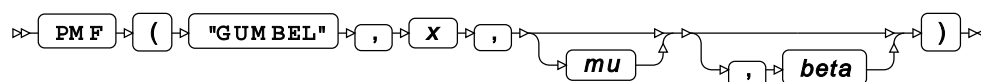
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

PMF – GUMBEL

Returns the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of PDF – GUMBEL.



The probability density function at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The probability density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: *beta* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
s1= PMF ("GUMBEL", 1, 0, 1);
PUT s1;
s2= PMF ("GUMBEL", 1, 0);
PUT s2;
s3= PMF ("GUMBEL", 1);
PUT s3;
RUN;
```

This produces the following output:

```
s1=0.25464638
s2=0.25464638
s3=0.25464638
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = PMF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.0605123465
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a much lower probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = PMF("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = PMF("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=0.0066926997  
s6=0.0066926997
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;  
s7 = PMF("GUMBEL", 1, 1, 0, 0);  
PUT s7=;  
s8 = PMF("GUMBEL", 1, , 0);  
PUT s8=;  
RUN;
```

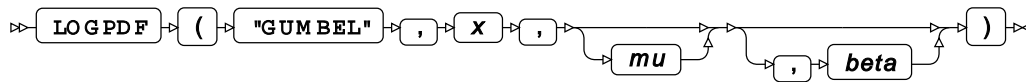
This produces the following output:

```
s7=.  
s8=.
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

LOGPDF – GUMBEL

Returns the natural logarithm of the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of LOGPMF – GUMBEL.



The probability density function at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The probability density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \quad \text{where } z = \frac{x - \mu}{\beta}$$

The natural logarithm of this is therefore:

$$f(x; \mu, \beta) = \log\left[\frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right)\right] \quad \text{where } z = \frac{x - \mu}{\beta}$$

which simplifies to:

$$f(x; \mu, \beta) = -\log\beta - (z + e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGPDF("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= LOGPDF("GUMBEL", 1, 0);  
PUT s2=;  
s3= LOGPDF("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-1.367879441  
s2=-1.367879441  
s3=-1.367879441
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda=1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same probability density.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGPDF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-2.804907861
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower natural logarithm of the probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = LOGPDF("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = LOGPDF("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=-5.006737947  
s6=-5.006737947
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = LOGPDF("GUMBEL", 1, 1, 0, 0);
PUT s7=;
s8 = LOGPDF("GUMBEL", 1, , 0);
PUT s8=;
RUN;
```

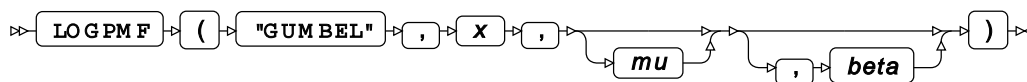
This produces the following output:

```
s7=.
s8=.
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

LOGPMF – GUMBEL

Returns the natural logarithm of the probability density at a point from a Gumbel distribution, with specified location and scale parameters. This function is an alias of LOGPDF – GUMBEL.



The probability density at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The probability density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right) \quad \text{where } z = \frac{x - \mu}{\beta}$$

The natural logarithm of this is therefore:

$$f(x; \mu, \beta) = \log\left[\frac{1}{\beta} \exp\left(-\left(z + e^{-z}\right)\right)\right] \quad \text{where } z = \frac{x - \mu}{\beta}$$

which simplifies to:

$$f(x; \mu, \beta) = -\log \beta - (z + e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: *beta* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGPMF("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= LOGPMF("GUMBEL", 1, 0);  
PUT s2=;  
s3= LOGPMF("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-1.367879441  
s2=-1.367879441  
s3=-1.367879441
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same natural logarithm of the probability density.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGPMF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-2.804907861
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower natural logarithm of the probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = LOGPMF("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = LOGPMF("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=-5.006737947  
s6=-5.006737947
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;  
s7 = LOGPMF("GUMBEL", 1, 1, 0, 0);  
PUT s7=;  
s8 = LOGPMF("GUMBEL", 1, , 0);  
PUT s8=;  
RUN;
```

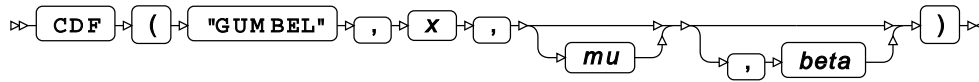
This produces the following output:

```
s7=.  
s8=.
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

CDF – GUMBEL

Returns the cumulative density at a point from a Gumbel distribution, with specified location and scale parameters.



The cumulative density function at a point x gives the likelihood that a randomly drawn value from the distribution is less than or equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The cumulative density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \exp(-e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= CDF ("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= CDF ("GUMBEL", 1, 0);  
PUT s2=;  
s3= CDF ("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=0.6922006276  
s2=0.6922006276  
s3=0.6922006276
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same cumulative density.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = CDF ("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.4289213437
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower cumulative density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = CDF ("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = CDF ("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=0.9932847021  
s6=0.9932847021
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the CDF from the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = CDF("GUMBEL", 1, 1, 0, 0);
PUT s7=;
s8 = CDF("GUMBEL", 1, , 0);
PUT s8=;
RUN;
```

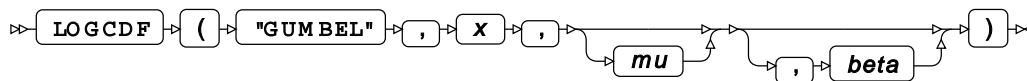
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

LOGCDF – GUMBEL

Returns the natural logarithm of the cumulative density at a point from a Gumbel distribution, with specified location and scale parameters.



The cumulative density function at a point x gives the likelihood that a randomly drawn value from the distribution is less than or equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The cumulative density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = \exp(-e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

The natural logarithm of this is therefore:

$$f(x; \mu, \beta) = -e^{-z} \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: *beta* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGCDF("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= LOGCDF("GUMBEL", 1, 0);  
PUT s2=;  
s3= LOGCDF("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-0.367879441  
s2=-0.367879441  
s3=-0.367879441
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGCDF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-0.846481725
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower natural logarithm of cumulative density at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
s5 = LOGCDF("GUMBEL", 5, 0, 1);
PUT s5=;
s6 = LOGCDF("GUMBEL", 2, -3, 1);
PUT s6=;
RUN;
```

This produces the following output:

```
s5=-0.006737947
s6=-0.006737947
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the CDF from the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = LOGCDF("GUMBEL", 1, 1, 0, 0);
PUT s7=;
s8 = LOGCDF("GUMBEL", 1, , 0);
PUT s8=;
RUN;
```

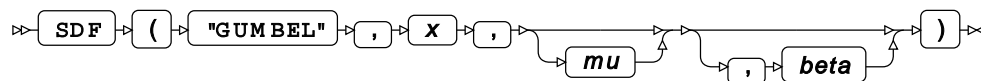
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

SDF – GUMBEL

Returns the survival density at a point from a Gumbel distribution, with specified location and scale parameters.



The survival density function at a point x gives the likelihood that a randomly drawn value from the distribution is greater than or equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The survival density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$
$$f(x; \mu, \beta) = 1 - \exp(-e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: *beta* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
s1= SDF("GUMBEL", 1, 0, 1);
PUT s1;
s2= SDF("GUMBEL", 1, 0);
PUT s2;
s3= SDF("GUMBEL", 1);
PUT s3;
RUN;
```

This produces the following output:

```
s1=0.3077993724
s2=0.3077993724
s3=0.3077993724
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same survival density.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = SDF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.5710786563
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a higher survival density at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = SDF("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = SDF("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=0.0067152979  
s6=0.0067152979
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the SDF from the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;  
s7 = SDF("GUMBEL", 1, 1, 0, 0);  
PUT s7=;  
s8 = SDF("GUMBEL", 1, , 0);  
PUT s8=;  
RUN;
```

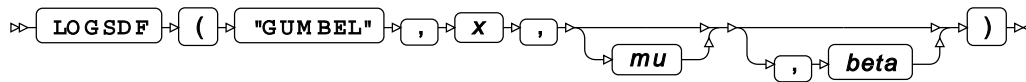
This produces the following output:

```
s7=.  
s8=.
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

LOGSDF – GUMBEL

Returns the natural logarithm of the survival density at a point from a Gumbel distribution, with specified location and scale parameters.



The survival density function at a point x gives the likelihood that a randomly drawn value from the distribution is greater than or equal to x .

You can optionally specify the distribution's location parameter μ (*mu*) and scale parameter β (*beta*). If these arguments are not specified, a standard Gumbel distribution is assumed, with $\mu = 0$ and $\beta = 1$. It is possible to specify μ (*mu*) without β (*beta*) (in which case, β is set to 1), but not β (*beta*) without μ (*mu*).

The survival density function at a point x of a Gumbel distribution with location parameter μ and scale parameter β is defined as follows:

$$\beta > 0$$

$$f(x; \mu, \beta) = 1 - \exp(-e^{-z}) \quad \text{where } z = \frac{x - \mu}{\beta}$$

The natural logarithm of this is therefore:

$$f(x; \mu, \beta) = \log(1 - \exp(-e^{-z})) \quad \text{where } z = \frac{x - \mu}{\beta}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival density.

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: $\beta > 0$

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGSDF("GUMBEL", 1, 0, 1);  
PUT s1=;  
s2= LOGSDF("GUMBEL", 1, 0);  
PUT s2=;  
s3= LOGSDF("GUMBEL", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-1.178307096  
s2=-1.178307096  
s3=-1.178307096
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\lambda = 1$ and $\theta = 0$ (a 'standard' Gumbel distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGSDF("GUMBEL", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-0.560228327
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a higher value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = LOGSDF("GUMBEL", 5, 0, 1);  
PUT s5=;  
s6 = LOGSDF("GUMBEL", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=-5.003367082  
s6=-5.003367082
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\lambda = 1$ and $\theta = 0$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the SDF from the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = LOGSDF("GUMBEL", 1, 1, 0, 0);
PUT s7=;
s8 = LOGSDF("GUMBEL", 1, , 0);
PUT s8=;
RUN;
```

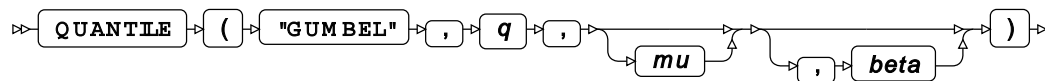
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

QUANTILE – GUMBEL

Returns the quantile of a Gumbel distribution with specified location and scale parameters, for a specified cumulative density.



The quantile for a cumulative density q gives a threshold value of x , below which randomly drawn values from the defined Gumbel distribution should occur q times.

You can optionally specify the distribution's location parameter μ (mu) and scale parameter β ($beta$). If these arguments are not specified, a standard Gumbel distribution is assumed, which has $\mu = 0$ and $\beta = 1$. It is possible to specify μ (mu) without β ($beta$) (in which case, β is set to 1), but not β ($beta$) without μ (mu).

The quantile function for a Gumbel distribution with location parameter μ and scale parameter β is defined as the inverse of the cumulative density function:

$$\beta > 0, 0 < q < 1$$

$$f(q; \mu, \beta) = \inf \{x : q \leq \text{CDF}(x, \mu, \beta)\}$$

Return type: Numeric

q

Type: Numeric

The cumulative probability density value for which to calculate the quantile.

Restriction: $0 < q < 1$

mu

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution on the x axis.

beta

Optional argument

Type: Numeric

The scale parameter. If used, *mu* must also be specified.

Restriction: $\beta \geq 0$

Basic example

```
DATA _NULL_;  
S1= QUANTILE("GUMBEL", 0.69220062755, 0, 1);  
PUT S1=;  
S2= QUANTILE("GUMBEL", 0.69220062755, 0);  
PUT S2=;  
S3= QUANTILE("GUMBEL", 0.69220062755);  
PUT S3=;  
RUN;
```

This produces the following output:

```
S1=1  
S2=1  
S3=1
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each specifying $q = 0.69220062755$, and in every case resolving to $\theta = 0$, and $\lambda = 1$; therefore returning the same value: $x = 1$.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = QUANTILE("GUMBEL", 0.428921343695, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=1
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This means that a smaller cumulative density is required to return an x value of 1 compared to the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
S5 = QUANTILE("GUMBEL", 0.993284702068, 0, 1);  
PUT S5=;  
S6 = QUANTILE("GUMBEL", 0.993284702068, -3, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=5  
S6=2
```

These two examples demonstrate the location parameter. The same cumulative density is input, but with the Gumbel distribution at two different locations, thus producing different values of x .

Example – invalid syntax

```
DATA _NULL_;  
S7 = QUANTILE("GUMBEL", 0.993284702068, 1, 0, 1, 0);  
PUT S7=;  
S8 = QUANTILE("GUMBEL", 0.993284702068, , 1, 0);  
PUT S8=;  
RUN;
```

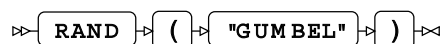
This produces the following output:

```
S7= .  
S8= .
```

Neither of these examples are valid; the first has too many arguments, and the second specifies λ without θ .

RAND – GUMBEL

Returns a random number from the Gumbel distribution.



```
>> RAND ( "GUMBEL" ) <<
```

The distribution is parameterised using a location of 0 and a scale of 1.

This function does not take any variable arguments.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

Example

In this example, a random number from the Gumbel distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("GUMBEL");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-0.851020115  
-0.367363323  
-1.019768582  
1.3707779555  
0.6873485682
```

Running the `DATA` step again produces the following output.

```
The random numbers are:  
-0.779538288  
-1.013083359  
-1.249802502  
1.5490383986  
-0.116005925
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(18);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("GUMBEL");  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:
2.1788597906
1.1880261682
-1.049847839
-0.185175736
0.0890908226
```

Running the `DATA` step again produces the same output.

Hypergeometric distribution

Functions for the Hypergeometric distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

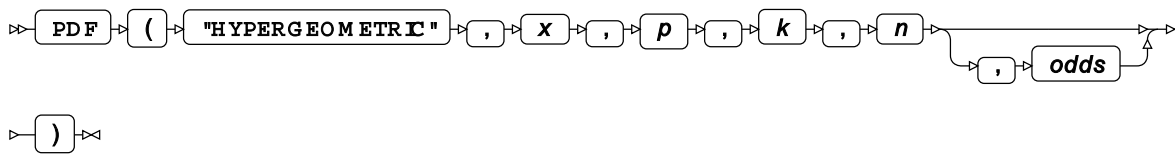
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – HYPERGEOMETRIC ↗	1007
Returns the probability density of the Hypergeometric distribution. This function is an alias of PMF – HYPERGEOMETRIC.	
PMF – HYPERGEOMETRIC ↗	1008
Returns the probability mass of the Hypergeometric distribution. This function is an alias of PDF – HYPERGEOMETRIC.	
LOGPDF – HYPERGEOMETRIC ↗	1008
Returns the natural logarithm of the probability density of the Hypergeometric distribution. This function is an alias of LOGPMF – HYPERGEOMETRIC.	
LOGPMF – HYPERGEOMETRIC ↗	1009
Returns the natural logarithm of the probability mass of the Hypergeometric distribution. This function is an alias of LOGPDF – HYPERGEOMETRIC.	
CDF – HYPERGEOMETRIC ↗	1010
Returns the cumulative density of the Hypergeometric distribution.	
PROBHYPR ↗	1011
LOGCDF – HYPERGEOMETRIC ↗	1011
Returns the natural logarithm of the cumulative density of the Hypergeometric distribution.	
SDF – HYPERGEOMETRIC ↗	1012
Returns the survival of the Hypergeometric distribution.	

LOGSDF – HYPERGEOMETRIC ↗	1013
Returns the natural logarithm of the survival of the Hypergeometric distribution.	
QUANTILE – HYPERGEOMETRIC ↗	1014
Returns the quantile of the Hypergeometric distribution.	
RAND – HYPER ↗	1014
Returns a random number from the Hypergeometric distribution based on the population size, number of draws, and number of successes.	

PDF – HYPERGEOMETRIC

Returns the probability density of the Hypergeometric distribution. This function is an alias of PMF – HYPERGEOMETRIC.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

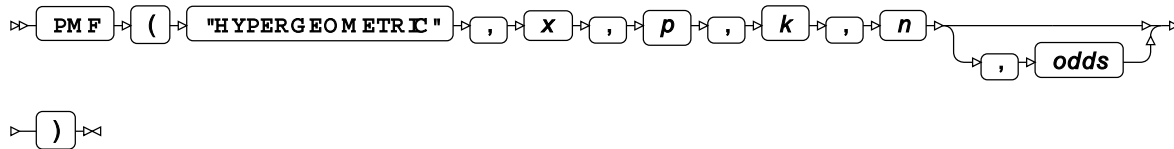
odds

Optional argument

Type: Numeric

PMF – HYPERGEOMETRIC

Returns the probability mass of the Hypergeometric distribution. This function is an alias of PDF – HYPERGEOMETRIC.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

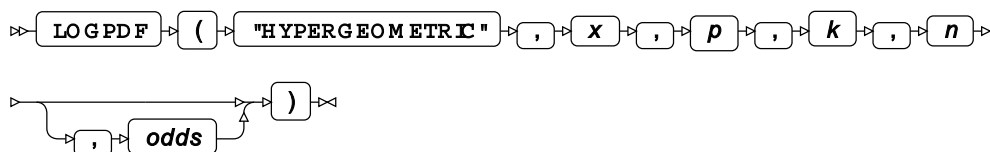
odds

Optional argument

Type: Numeric

LOGPDF – HYPERGEOMETRIC

Returns the natural logarithm of the probability density of the Hypergeometric distribution. This function is an alias of LOGPMF – HYPERGEOMETRIC.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

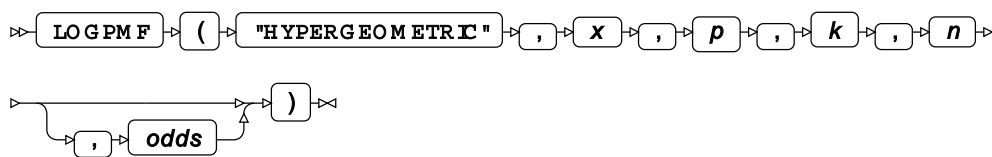
odds

Optional argument

Type: Numeric

LOGPMF – HYPERGEOMETRIC

Returns the natural logarithm of the probability mass of the Hypergeometric distribution. This function is an alias of LOGPDF – HYPERGEOMETRIC.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

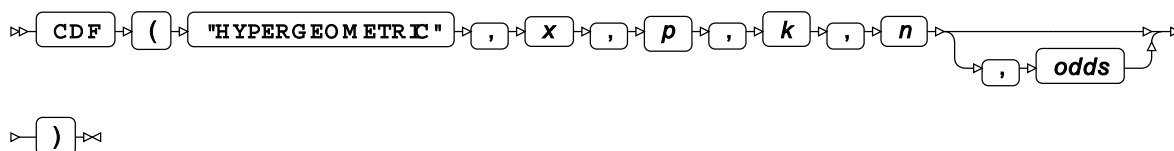
odds

Optional argument

Type: Numeric

CDF – HYPERGEOMETRIC

Returns the cumulative density of the Hypergeometric distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

p

Type: Numeric

k

Type: Numeric

n

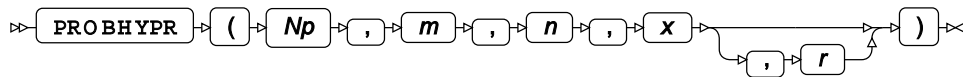
Type: Numeric

odds

Optional argument

Type: Numeric

PROBHYP



Return type: Numeric

Np

Type: Numeric

m

Type: Numeric

n

Type: Numeric

x

Type: Numeric

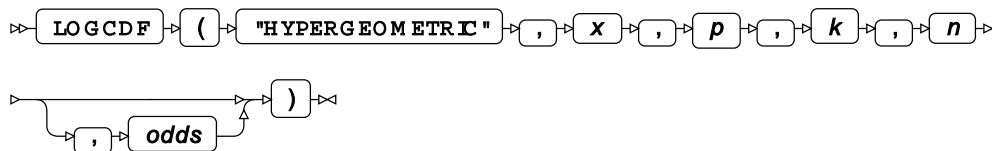
r

Optional argument

Type: Numeric

LOGCDF – HYPERGEOMETRIC

Returns the natural logarithm of the cumulative density of the Hypergeometric distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

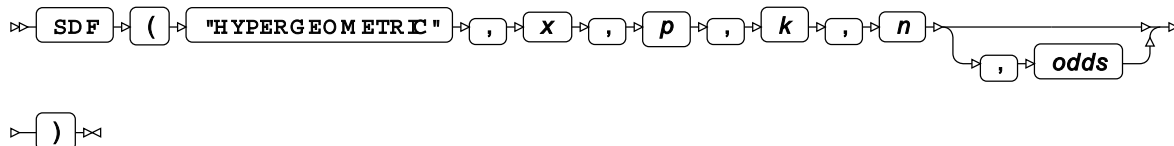
odds

Optional argument

Type: Numeric

SDF – HYPERGEOMETRIC

Returns the survival of the Hypergeometric distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

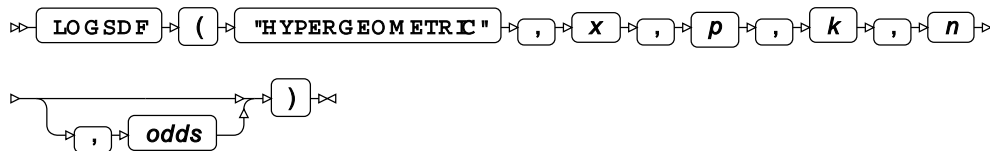
odds

Optional argument

Type: Numeric

LOGSDF – HYPERGEOMETRIC

Returns the natural logarithm of the survival of the Hypergeometric distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

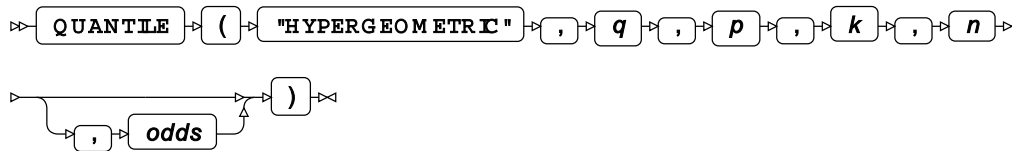
odds

Optional argument

Type: Numeric

QUANTILE – HYPERGEOMETRIC

Returns the quantile of the Hypergeometric distribution.



Return type: Numeric

q

Type: Numeric

p

Type: Numeric

k

Type: Numeric

n

Type: Numeric

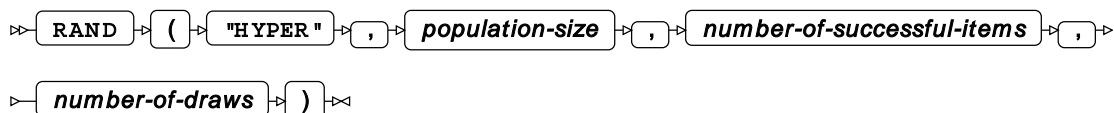
odds

Optional argument

Type: Numeric

RAND – HYPER

Returns a random number from the Hypergeometric distribution based on the population size, number of draws, and number of successes.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS)* - Special issue on uniform random number generation 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

population-size

Type: Numeric

The size of the population from which draw are made.

number-of-successful-items

Type: Numeric

The number of draws that meet the criterion for success.

number-of-draws

Type: Numeric

The total number of draws from the population.

Example

In this example, a random number from the Hypergeometric distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("HYPER", 1000, 450, 600);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
282  
272  
285  
273  
271
```

Running the DATA step again produces the following output.

```
The random numbers are:
258
266
277
277
274
```

However, if you first use CALL STREAMINIT to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(18);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("HYPER", 1000, 450, 600);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
264
279
283
271
270
```

Running the DATA step again produces the same output.

Inverse Gaussian distribution

Functions for the Inverse Gaussian distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – IGAUSS [↗](#)..... 1018

$$\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

Returns the value of the probability density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of PDF – IGAUSS, PDF – WALD and PMF – WALD.

PMF – IGAUSS [↗](#)..... 1020

$$\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)$$

Returns the value of the probability mass function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of PDF – IGAUSS, PDF – WALD and PMF – WALD.

LOGPDF – IGAUSS [↗](#)..... 1023

$$\log\left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)\right]$$

Returns the value of the natural logarithm of the probability density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGPMF – IGAUSS, LOGPDF – WALD and LOGPMF – WALD.

LOGPMF – IGAUSS [↗](#)..... 1025

$$\log\left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)\right]$$

Returns the value of the natural logarithm of the probability mass function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGPDF – IGAUSS, LOGPDF – WALD and LOGPMF – WALD.

CDF – IGAUSS [↗](#)..... 1028

$$\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Returns the value of the cumulative density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of CDF – WALD.

LOGCDF – IGAUSS [↗](#)..... 1030

$$\log\left[\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt\right]$$

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGCDF – WALD.

SDF – IGAUSS [↗](#)..... 1033

$$1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Returns the value of the survival function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of SDF – WALD.

LOGSDF – IGAUSS [↗](#)..... 1035

$$\log \left[1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \frac{1}{t^3} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt \right]$$

Returns the value of the natural logarithm of the survival function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGSDF – WALD.

QUANTILE – IGAUSS [↗](#)..... 1037

$$\inf \{x: q \leq \text{CDF}(x; \lambda, \mu)\}$$

Returns the value of the quantile function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of QUANTILE – WALD.

DEVIANCE – IGAUSS [↗](#)..... 1040

$$(x - \mu)^2 / x\mu^2$$

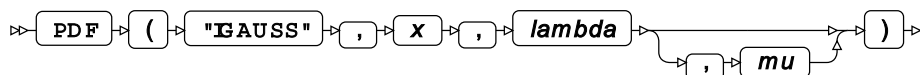
Returns the deviance of the Inverse Gaussian distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – WALD.

RAND – INVERSE GAUSSIAN [↗](#)..... 1042

Returns a random number from the Inverse Gaussian distribution based on the mean and shape.

PDF – IGAUSS

Returns the value of the probability density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of PDF – IGAUSS, PDF – WALD and PMF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 \text{ return } 0 \\ \text{if } x > 0 \text{ return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

lambda

Type: Numeric

Restriction: $\lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = PDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = PDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = PDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

s5 = PDF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.6429310692
s2=0.0562223942
s3=0.9678828981
s4=0.0144476475
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, PDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = PDF("IGAUSS", 0.5);
PUT s2=;

s3 = PDF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = PDF("IGAUSS", 0.5, 1, 0);
PUT s4=;

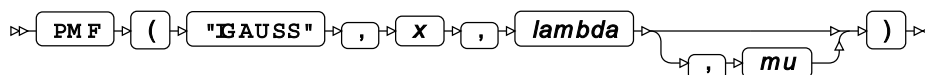
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

PMF – IGAUSS

Returns the value of the probability mass function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of PDF – IGAUSS, PDF – WALD and PMF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point x is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$
$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

λ

Type: Numeric

Restriction: $\lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

μ

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PMF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = PMF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = PMF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = PMF("IGAUSS", 3.5, 2, 1);
PUT s4=;

s5 = PMF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.6429310692
s2=0.0562223942
s3=0.9678828981
s4=0.0144476475
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, PMF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = PMF("IGAUSS", 0.5);
PUT s2=;

s3 = PMF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = PMF("IGAUSS", 0.5, 1, 0);
PUT s4=;

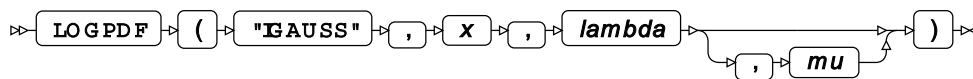
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGPDF – IGAUSS

Returns the value of the natural logarithm of the probability density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGPMF – IGAUSS, LOGPDF – WALD and LOGPMF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right) \right]$$

Return type: Numeric

x

Type: Numeric

Restriction: $x > 0$

The point at which to calculate the natural logarithm of the probability density.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = LOGPDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = LOGPDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = LOGPDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-0.441717762
s2=-2.878440129
s3=-0.032644172
s4=-4.237223681
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGPDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF("IGAUSS", 0.5, 1, 2, 3);
  PUT s1=;

  s2 = LOGPDF("IGAUSS", 0.5);
  PUT s2=;

  s3 = LOGPDF("IGAUSS", 0.5, 0, 2);
  PUT s3=;

  s4 = LOGPDF("IGAUSS", 0.5, 1, 0);
  PUT s4=;

  s5 = LOGPDF("IGAUSS", -0.5, 1, 2);
  PUT s5=;

RUN;
```

This produces the following output:

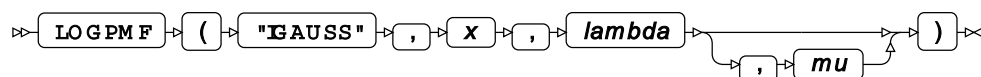
```
s1=.
s2=.
s3=.
s4=.
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

LOGPMF – IGAUSS

Returns the value of the natural logarithm of the probability mass function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGPDF – IGAUSS, LOGPDF – WALD and LOGPMF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right) \right]$$

Return type: Numeric

x

Type: Numeric

Restriction: $x > 0$

The point at which to calculate the natural logarithm of the probability mass.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPMF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPMF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = LOGPMF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = LOGPMF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = LOGPMF("IGAUSS", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-0.441717762
s2=-2.878440129
s3=-0.032644172
s4=-4.237223681
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGPMF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;
s1 = LOGPMF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = LOGPMF("IGAUSS", 0.5);
PUT s2=;

s3 = LOGPMF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = LOGPMF("IGAUSS", 0.5, 1, 0);
PUT s4=;

s5 = LOGPMF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

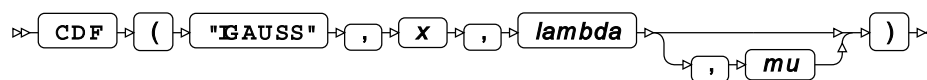
```
s1=.
s2=.
s3=.
s4=.
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

CDF – IGAUSS

Returns the value of the cumulative density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of CDF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

lambda**Type:** Numeric**Restriction:** $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric**Default:** 1**Restriction:** $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, CDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = CDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = CDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = CDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

s5 = CDF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.2492117733
s2=0.8481757552
s3=0.2323571892
s4=0.9888921338
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, CDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = CDF("IGAUSS", 0.5);
PUT s2=;

s3 = CDF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = CDF("IGAUSS", 0.5, 1, 0);
PUT s4=;

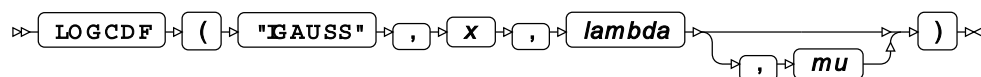
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGCDF – IGAUSS

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGCDF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt \right]$$

Return type: Numeric

x

Type: Numeric

Restriction: $x > 0$

The point at which to calculate the natural logarithm of the cumulative density.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGCDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = LOGCDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = LOGCDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = LOGCDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-1.389452249  
s2=-0.164667406  
s3=-1.459479483  
s4=-0.011170019
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGCDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;  
  
s1 = LOGCDF("IGAUSS", 0.5, 1, 2, 3);  
PUT s1=;  
  
s2 = LOGCDF("IGAUSS", 0.5);  
PUT s2=;  
  
s3 = LOGCDF("IGAUSS", 0.5, 0, 2);  
PUT s3=;  
  
s4 = LOGCDF("IGAUSS", 0.5, 1, 0);  
PUT s4=;  
  
s5 = LOGCDF("IGAUSS", -0.5, 1, 2);  
PUT s5=;  
  
RUN;
```

This produces the following output:

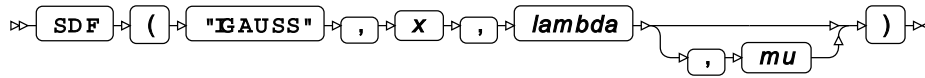
```
s1=.  
s2=.  
s3=.  
s4=.  
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

SDF – IGAUSS

Returns the value of the survival function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of SDF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 1 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = 1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the value of the survival function.

lambda

Type: Numeric

Restriction: $\lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, SDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = SDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = SDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = SDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

s5 = SDF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.7507882267
s2=0.1518242448
s3=0.7676428108
s4=0.0111078662
s5=1
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, SDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = SDF("IGAUSS", 0.5);
PUT s2=;

s3 = SDF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = SDF("IGAUSS", 0.5, 1, 0);
PUT s4=;

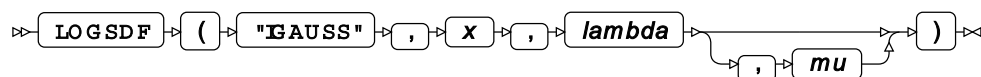
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGSDF – IGAUSS

Returns the value of the natural logarithm of the survival function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of LOGSDF – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \log \left[1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \frac{1}{t^3} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGSDF – IGAUSS is called for various Inverse Gaussian distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGSDF("IGAUSS", 0.5, 1, 2);
PUT s1=;
s2 = LOGSDF("IGAUSS", 3.5, 1, 2);
PUT s2=;

s3 = LOGSDF("IGAUSS", 0.5, 2, 1);
PUT s3=;
s4 = LOGSDF("IGAUSS", 3.5, 2, 1);
PUT s4=;

s5 = LOGSDF("IGAUSS", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=-0.286631655
s2=-1.885031712
s3=-0.264430744
s4=-4.500101754
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, LOGSDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGSDF("IGAUSS", 0.5, 1, 2, 3);
PUT s1=;

s2 = LOGSDF("IGAUSS", 0.5);
PUT s2=;

s3 = LOGSDF("IGAUSS", 0.5, 0, 2);
PUT s3=;

s4 = LOGSDF("IGAUSS", 0.5, 1, 0);
PUT s4=;

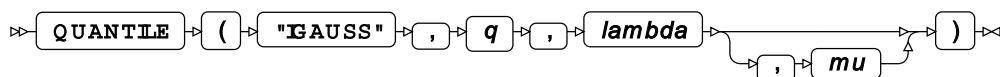
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

QUANTILE – IGAUSS

Returns the value of the quantile function at a specified point for the Inverse Gaussian distribution with the specified shape and mean. This function is an alias of QUANTILE – WALD.



The Inverse Gaussian distribution is also known as the Wald distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} 0 < q < 1 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point q is

$$f(q; \lambda, \mu) = \inf \{x: q \leq \text{CDF}(x; \lambda, \mu)\}$$

where

- $x \in \mathbb{R}$
- $\inf\{x\}$ (*infinium*) is the greatest lower bound of x
- λ is the shape parameter
- μ is the mean
- $\text{CDF}(x; \lambda, \mu)$ is the cumulative density function

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, QUANTILE – IGAUSS is called twice, with two different probability values from the same distribution. The returned values are then passed to CDF – IGAUSS for the same distribution. The results are written to the log.

```
DATA _NULL_;

  s1 = QUANTILE("IGAUSS", 0.25, 1, 2);
  PUT s1=;
  s2 = QUANTILE("IGAUSS", 0.75, 1, 2);
  PUT s2=;

  s3 = CDF("IGAUSS", s1, 1, 2);
  PUT s3=;
  s4 = CDF("IGAUSS", s2, 1, 2);
  PUT s4=;

RUN;
```

This produces the following output:

```
s1=0.5012268362
s2=2.2841739815
s3=0.25
s4=0.75
```

As QUANTILE – IGAUSS and CDF – IGAUSS are inverse functions, the values returned from CDF – IGAUSS are the same as those originally passed to QUANTILE – IGAUSS.

Argument errors

In this example, CDF – IGAUSS is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = QUANTILE("IGAUSS", 0.25, 1, 2, 3);
  PUT s1=;

  s2 = QUANTILE("IGAUSS", 0.25);
  PUT s2=;

  s3 = QUANTILE("IGAUSS", 0.25, 0, 2);
  PUT s3=;

  s4 = QUANTILE("IGAUSS", 0.25, 1, 0);
  PUT s4=;

  s5 = QUANTILE("IGAUSS", 0, 1, 0);
  PUT s5=;

  s6 = QUANTILE("IGAUSS", -1, 1, 0);
  PUT s6=;

RUN;
```

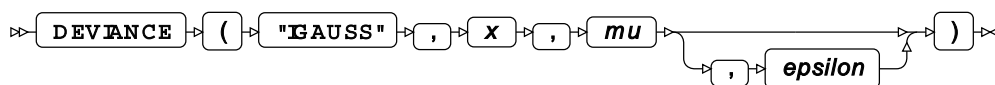
All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth and sixth examples specify invalid values for *q*.

DEVIANCE – IGAUSS

Returns the deviance of the Inverse Gaussian distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – WALD.



Keyword `IGAUSS` is an alias of `WALD`, see [DEVIANCE – WALD](#) (page 1359).

Calculates the deviance, or goodness of fit, for the generalised linear model of the Inverse Gaussian distribution at a nonnegative point x based on the distribution mean μ (mu). An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The distribution mean is then adjusted so that $\mu \geq \varepsilon$:

$$\text{if } \mu < \varepsilon \text{ set } \mu = \varepsilon$$

If $x \geq 0$, it is adjusted so that $x \geq \varepsilon$:

$$\text{if } 0 \leq x < \varepsilon \text{ set } x = \varepsilon$$

These adjusted values of μ and x are used in the subsequent calculation of the deviance.

$$(x - \mu)^2 / x\mu^2$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

mu

Type: Numeric

The distribution mean.

Expected: $\mu > 0$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$

Expected: $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Examples – applying correction to the distribution mean

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
g1 = DEVIANCE("IGAUSS", 0.01, 0.0007, 0.0005);  
PUT g1=;  
g2 = DEVIANCE("IGAUSS", 0.01, 0.0007, 0.0010);  
PUT g2=;  
g3 = DEVIANCE("IGAUSS", 0.01, 0.0007, 0.0015);  
PUT g3=;  
g4 = DEVIANCE("IGAUSS", 0.01, 0.0007          );  
PUT g4=;  
RUN;
```

This produces the following output:

```
g1=17651.020408  
g2=8100  
g3=3211.1111111  
g4=17651.020408
```

The value of the distribution mean is not corrected in the first example because $\mu > \varepsilon$. However, this condition does not hold in the second and third example, and correction is applied: $\mu = \varepsilon$. This corrected value is used for calculation, yielding different results.

In the fourth example the ε parameter is omitted, so the default value of $\varepsilon = 10^{-12}$ is used. Here, as in the first example, $\mu > \varepsilon$, so no correction is required.

Examples – inverted scaling

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = DEVIANCE("IGAUSS", 0.01, 0.07);
  PUT g1=;
  g2 = DEVIANCE("IGAUSS", 0.1 , 0.7 );
  PUT g2=;
  g3 = DEVIANCE("IGAUSS", 1 , 7 );
  PUT g3=;
RUN;
```

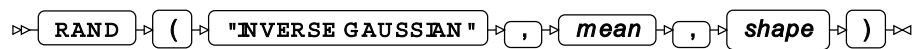
This produces the following output:

```
g1=73.469387755
g2=7.3469387755
g3=0.7346938776
```

For the Inverse Gaussian distribution, when both the point of measurement x and the distribution mean μ are scaled with the same factor, the deviance is scaled with the inverse of this factor.

RAND – INVERSE GAUSSIAN

Returns a random number from the Inverse Gaussian distribution based on the mean and shape.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

mean

Type: Numeric

The mean of the distribution.

shape

Type: Numeric

The shape for the distribution.

Example

In this example, a random number from the Inverse Gaussian distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("INVERSE GAUSSIAN", 50, 4);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
19.216894327  
88.842156238  
6.6845410249  
5.3918984993  
1.6023162835
```

Running the DATA step again produces the following output.

```
The random numbers are:  
1.5702146415  
12.69443758  
31.972435895  
2.4117557127  
5.2739838598
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(12);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("INVERSE GAUSSIAN", 50, 4);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
5.0841749568  
1.2554045238  
0.4110029909  
4.2476360314  
10.979380548
```

Running the DATA step again produces the same output.

Johnson SB distribution

Functions for the Johnson SB distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – JOHNSON SB [↗](#)..... 1045

$$\frac{\delta \exp\left(-\frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2\right)}{\lambda \sqrt{2\pi} z(1-z)} \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the probability density of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of PMF – JOHNSON SB.

PMF – JOHNSON SB [↗](#)..... 1047

$$\frac{\delta \exp\left(-\frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2\right)}{\lambda \sqrt{2\pi} z(1-z)} \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the probability mass of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of PDF – JOHNSON SB.

LOGPDF – JOHNSON SB [↗](#)..... 1050

$$\log \frac{\delta}{\lambda \sqrt{2\pi} z(1-z)} - \frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2 \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the natural logarithm of the probability density of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of LOGPMF – JOHNSON SB.

LOGPMF – JOHNSON SB [↗](#)..... 1052

$$\log \frac{\delta}{\lambda \sqrt{2\pi} z(1-z)} - \frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2 \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the natural logarithm of the probability mass of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of LOGPDF – JOHNSON SB.

CDF – JOHNSON SB [↗](#)..... 1054

$$\Phi\left(\gamma + \delta \log \frac{z}{1-z}\right) \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the cumulative density of the Johnson SB distribution based on the shape parameters, scale and location.

LOGCDF – JOHNSON SB [↗](#)..... 1056

$$\log \Phi\left(\gamma + \delta \log \frac{z}{1-z}\right) \quad \text{where } z = \frac{x-\xi}{\lambda}$$

Returns the natural logarithm of the cumulative density of the Johnson SB distribution based on the shape parameters, scale and location.

SDF – JOHNSON SB [↗](#)..... 1059

$$1 - \Phi \left(\gamma + \delta \log \frac{z}{1-z} \right) \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the survival of the Johnson SB distribution based on the shape parameters, scale and location.

LOGSDF – JOHNSON SB [↗](#)..... 1061

$$\log \left[1 - \Phi \left(\gamma + \delta \log \frac{z}{1-z} \right) \right] \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the natural logarithm of the survival of the Johnson SB distribution based on the shape parameters, scale and location.

QUANTILE – JOHNSON SB [↗](#)..... 1063

$$f(q) = \frac{\lambda \exp(\varphi)}{1 + \exp(\varphi)} + \xi \quad \text{where } \varphi = \frac{\Phi^{-1}(q) - \gamma}{\delta}$$

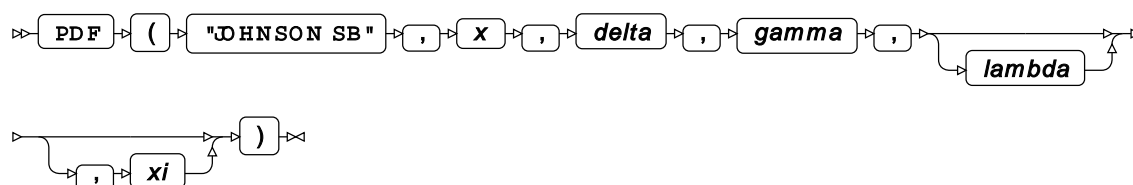
Returns the quantile of the Johnson SB distribution based on the shape parameters, scale and location.

RAND – JOHNSON SB [↗](#)..... 1066

Returns a random number from the Johnson SB distribution based on the shape parameters.

PDF – JOHNSON SB

Returns the probability density of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of PMF – JOHNSON SB.



Calculates the probability density function for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$\begin{cases} \text{if } x \leq \xi & \text{return } 0 \\ \text{if } x \geq \xi + \lambda & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = \frac{\delta \exp\left(-\frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2\right)}{\lambda \sqrt{2\pi} z(1-z)} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If xi is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the probability density of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = PDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = PDF ("JOHNSON SB", 0.1, 1, -1, 1, 0);
  PUT s3=;
  s4 = PDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = PDF ("JOHNSON SB", 0.1, 1, 0, 1, -0.5);
  PUT s5=;
  s6 = PDF ("JOHNSON SB", 0.1, 0, 0, 1, -0.5);
  PUT s6=;
RUN;
```

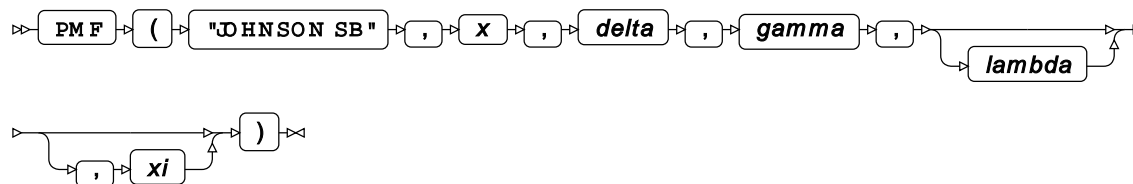
This produces the following output:

```
s1=0.3965747044
s2=0.0005679726
s3=0.0267260797
s4=0.0550275892
s5=1.5310853232
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

PMF – JOHNSON SB

Returns the probability mass of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of PDF – JOHNSON SB.



Calculates the probability mass function for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$\begin{cases} \text{if } x \leq \xi & \text{return } 0 \\ \text{if } x \geq \xi + \lambda & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = \frac{\delta \exp\left(-\frac{1}{2}\left(\gamma + \delta \log \frac{z}{1-z}\right)^2\right)}{\lambda \sqrt{2\pi} z(1-z)} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the probability mass of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = PMF ("JOHNSON SB", 0.1, 1, 0, 1, 0);  
  PUT s1=;  
  s2 = PMF ("JOHNSON SB", 0.1, 2, 0, 1, 0);  
  PUT s2=;  
  s3 = PMF ("JOHNSON SB", 0.1, 1,-1, 1, 0);  
  PUT s3=;  
  s4 = PMF ("JOHNSON SB", 0.1, 1, 0, 2, 0);  
  PUT s4=;  
  s5 = PMF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);  
  PUT s5=;  
  s6 = PMF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);  
  PUT s6=;  
RUN;
```

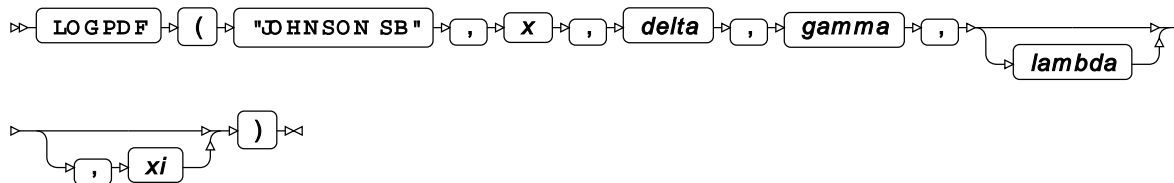
This produces the following output:

```
s1=0.3965747044  
s2=0.0005679726  
s3=0.0267260797  
s4=0.0550275892  
s5=1.5310853232  
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGPDF – JOHNSON SB

Returns the natural logarithm of the probability density of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of LOGPMF – JOHNSON SB.



Calculates the natural logarithm of the probability density for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0, \xi < x < \xi + \lambda$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \frac{\delta}{\lambda \sqrt{2\pi} z(1-z)} - \frac{1}{2} \left(\gamma + \delta \log \frac{z}{1-z} \right)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

Restriction: $\xi < x < \xi + \lambda$

If the argument is out of range, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGPDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGPDF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGPDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGPDF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = LOGPDF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

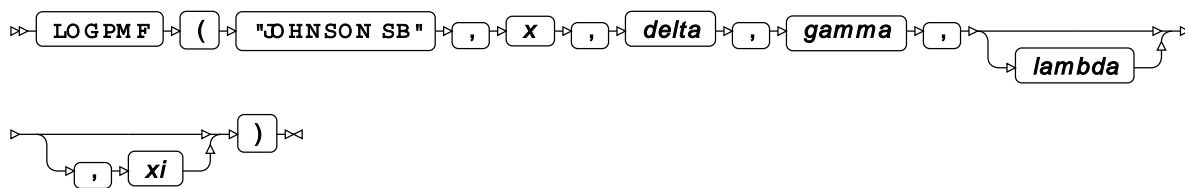
This produces the following output:

```
s1=-0.924890846
s2=-7.47343743
s3=-3.622115424
s4=-2.899920597
s5=0.4259768455
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGPMF – JOHNSON SB

Returns the natural logarithm of the probability mass of the Johnson SB distribution based on the shape parameters, scale and location. This function is an alias of LOGPDF – JOHNSON SB.



Calculates the natural logarithm of the probability mass function for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0, \xi < x < \xi + \lambda$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \frac{\delta}{\lambda \sqrt{2\pi} z(1-z)} - \frac{1}{2} \left(\gamma + \delta \log \frac{z}{1-z} \right)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

Restriction: $\xi < x < \xi + \lambda$

If the argument is out of range, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGPMF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGPMF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGPMF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGPMF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = LOGPMF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

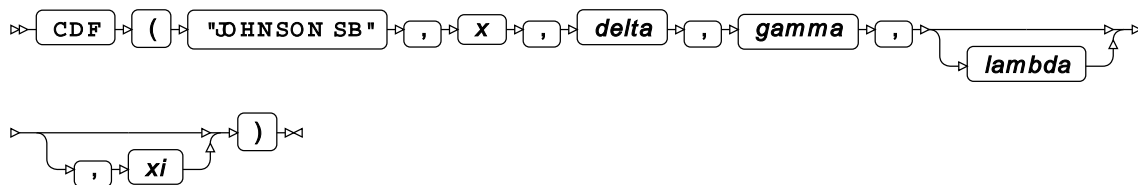
This produces the following output:

```
s1=-0.924890846
s2=-7.47343743
s3=-3.622115424
s4=-2.899920597
s5=0.4259768455
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

CDF – JOHNSON SB

Returns the cumulative density of the Johnson SB distribution based on the shape parameters, scale and location.



Calculates the cumulative density function for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$\begin{cases} \text{if } x \leq \xi & \text{return } 0 \\ \text{if } x \geq \xi + \lambda & \text{return } 1 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = \Phi\left(\gamma + \delta \log \frac{z}{1-z}\right) \quad \text{where } z = \frac{x - \xi}{\lambda}$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the cumulative density of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = CDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = CDF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = CDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = CDF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = CDF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

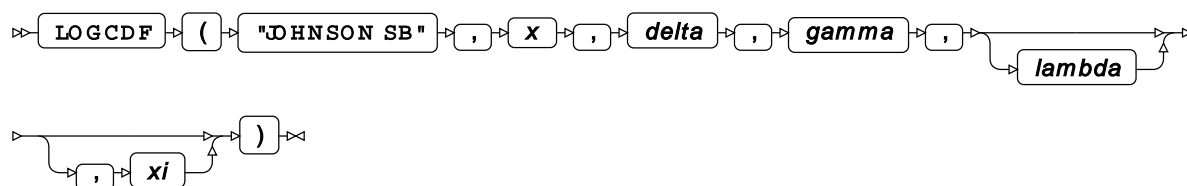
This produces the following output:

```
s1=0.0140022056
s2=5.5526996E-6
s3=0.0006937843
s4=0.0016177037
s5=0.6574321695
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGCDF – JOHNSON SB

Returns the natural logarithm of the cumulative density of the Johnson SB distribution based on the shape parameters, scale and location.



Calculates the natural logarithm of the cumulative density function for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0, x > \xi$$

$$\begin{cases} \text{if } x \geq \xi + \lambda & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \Phi \left(\gamma + \delta \log \frac{z}{1-z} \right) \quad \text{where } z = \frac{x - \xi}{\lambda}$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see *CDF – NORMAL* [↗](#) (page 1187) and *ERF* [↗](#) (page 1830).

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: $x > \xi$

If the argument is out of range, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If xi is specified, then $lambda$ must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If xi is specified, then $lambda$ must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGCDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGCDF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGCDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGCDF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = LOGCDF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

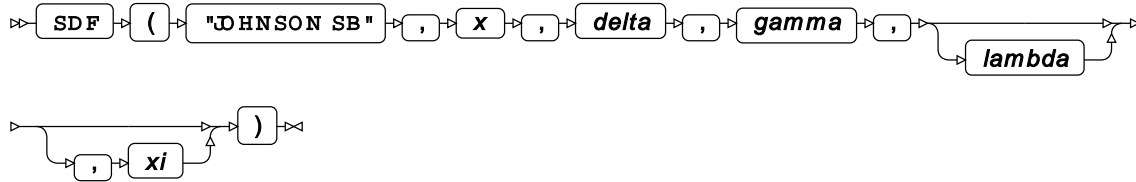
This produces the following output:

```
s1=-4.268540421
s2=-12.10122634
s3=-7.273349521
s4=-6.426747606
s5=-0.419413685
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

SDF – JOHNSON SB

Returns the survival of the Johnson SB distribution based on the shape parameters, scale and location.



Calculates the survival, or the complement to the cumulative density function, for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$\begin{cases} \text{if } x \leq \xi & \text{return } 1 \\ \text{if } x \geq \xi + \lambda & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = 1 - \Phi\left(\gamma + \delta \log \frac{z}{1-z}\right) \quad \text{where } z = \frac{x - \xi}{\lambda}$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the survival of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = SDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = SDF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = SDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = SDF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = SDF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

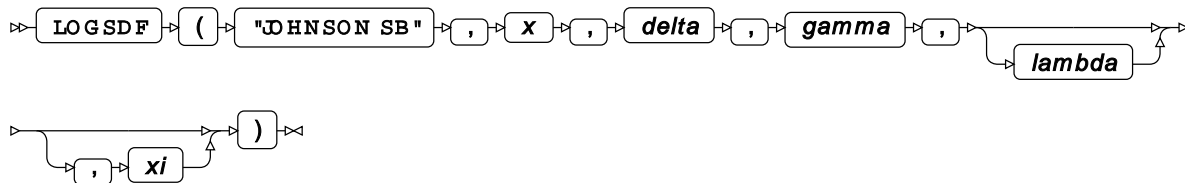
This produces the following output:

```
s1=0.9859977944
s2=0.9999944473
s3=0.9993062157
s4=0.9983822963
s5=0.3425678305
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGSDF – JOHNSON SB

Returns the natural logarithm of the survival of the Johnson SB distribution based on the shape parameters, scale and location.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Johnson SB distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0, x < \xi + \lambda$$

$$\begin{cases} \text{if } x \leq \xi & \text{return } 0 \\ \text{otherwise} & \text{return } f(x; \delta, \gamma, \lambda, \xi) \end{cases}$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \left[1 - \Phi \left(\gamma + \delta \log \frac{z}{1-z} \right) \right] \quad \text{where } z = \frac{x - \xi}{\lambda}$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

Restriction: $x < \xi + \lambda$

If the argument is out of range, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF ("JOHNSON SB", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGSDF ("JOHNSON SB", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGSDF ("JOHNSON SB", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGSDF ("JOHNSON SB", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGSDF ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);
  PUT s5=;
  s6 = LOGSDF ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);
  PUT s6=;
RUN;
```

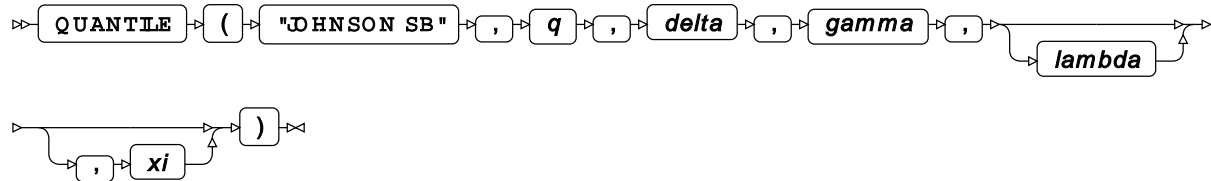
This produces the following output:

```
s1=-0.014101161
s2=-5.552715E-6
s3=-0.000694025
s4=-0.001619014
s5=-1.071285596
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

QUANTILE – JOHNSON SB

Returns the quantile of the Johnson SB distribution based on the shape parameters, scale and location.



Calculates the quantile x , or the inverse of the cumulative density function, for the Johnson SB distribution for probability value q based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$0 < q < 1, \delta > 0, \lambda > 0$$

$$f(q) = \frac{\lambda \exp(\varphi)}{1 + \exp(\varphi)} + \xi \quad \text{where} \quad \varphi = \frac{\Phi^{-1}(q) - \gamma}{\delta}$$

$$\Phi^{-1}(q) = \sqrt{2} \operatorname{erf}^{-1}(2q - 1)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi^{-1}(q)$ is the probit function, or the quantile function of the standard Normal distribution, and $\operatorname{erf}^{-1}(q)$ is the inverse of the error function, see *PROBIT* [↗](#) (page 1200) and *ERF* [↗](#) (page 1830).

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If xi is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

 xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If xi is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the quantile of the Johnson SB distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = QUANTILE ("JOHNSON SB", 0.1, 1, 0, 1, 0);  
  PUT s1=;  
  s2 = QUANTILE ("JOHNSON SB", 0.1, 2, 0, 1, 0);  
  PUT s2=;  
  s3 = QUANTILE ("JOHNSON SB", 0.1, 1,-1, 1, 0);  
  PUT s3=;  
  s4 = QUANTILE ("JOHNSON SB", 0.1, 1, 0, 2, 0);  
  PUT s4=;  
  s5 = QUANTILE ("JOHNSON SB", 0.1, 1, 0, 1,-0.5);  
  PUT s5=;  
  s6 = QUANTILE ("JOHNSON SB", 0.1, 0, 0, 1,-0.5);  
  PUT s6=;  
RUN;
```

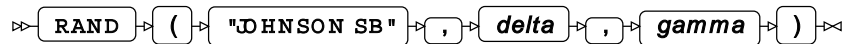
This produces the following output:

```
s1=0.2172862285  
s2=0.3450711937  
s3=0.4300734302  
s4=0.4345724571  
s5=-0.282713771  
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

RAND – JOHNSON SB

Returns a random number from the Johnson SB distribution based on the shape parameters.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

The Johnson SB distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

The return value is positive.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, an error is returned.

gamma

Type: Numeric

The second shape parameter.

Example

In this example, a random number from the Johnson SB distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("JOHNSON SB", 10,5);
    PUT result;
  END;
RUN;
```


This produces the following output:

```
The random numbers are:
0.3855994315
0.4378382275
0.3666275822
0.3726487926
0.3698356082
```

Running the DATA step again produces the following output.

```
The random numbers are:
0.3816704915
0.3473201428
0.372621443
0.3732903865
0.4076553479
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(18);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("JOHNSON SB", 10,5);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.389370907
0.3846661629
0.3702455951
0.3742395847
0.3672945964
```

Running the DATA step again produces the same output.

Johnson SU distribution

Functions for the Johnson SU distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – JOHNSON SU [↗](#)..... 1069

$$\frac{\delta \exp \left(-\frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \right)}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the probability density of the Johnson SU distribution, based on the shape parameters, scale and location. This function is an alias of PMF – JOHNSON SU.

PMF – JOHNSON SU [↗](#)..... 1071

$$\frac{\delta \exp \left(-\frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \right)}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the probability mass of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of PDF – JOHNSON SU.

LOGPDF – JOHNSON SU [↗](#)..... 1073

$$\log \frac{\delta}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} - \frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the natural logarithm of the probability density of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of LOGPMF – JOHNSON SU.

LOGPMF – JOHNSON SU [↗](#)..... 1075

$$\log \frac{\delta}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} - \frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

Returns the natural logarithm of the probability mass of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of LOGPDF – JOHNSON SU.

CDF – JOHNSON SU [↗](#)..... 1078

$$\Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right)$$

Returns the cumulative density of the Johnson SU distribution based on the shape parameters, scale and location.

LOGCDF – JOHNSON SU [↗](#)..... 1080

$$\log \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right)$$

Returns the natural logarithm of the cumulative density of the Johnson SU distribution based on the shape parameters, scale and location.

SDF – JOHNSON SU [↗](#)..... 1082

$$1 - \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right)$$

Returns the survival of the Johnson SU distribution based on the shape parameters, scale and location.

LOGSDF – JOHNSON SU [↗](#)..... 1085

$$\log \left[1 - \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right) \right]$$

Returns the natural logarithm of the survival of the Johnson SU distribution based on the shape parameters, scale and location.

QUANTILE – JOHNSON SU [🔗](#)..... 1087

$$f(q) = \lambda \sinh \frac{\Phi^{-1}(q) - \gamma}{\delta} + \xi$$

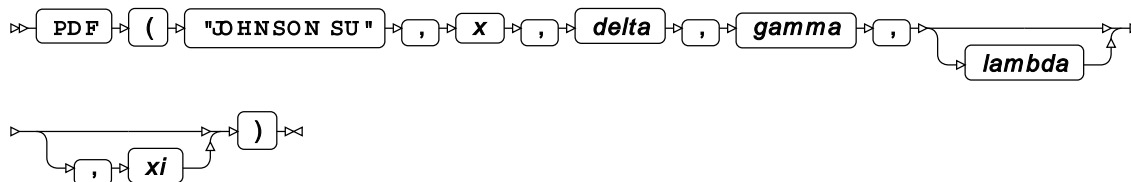
Returns the quantile of the Johnson SU distribution based on the shape parameters, scale and location.

RAND – JOHNSON SU [🔗](#)..... 1089

Returns a random number from the Johnson SU distribution based on the shape parameters.

PDF – JOHNSON SU

Returns the probability density of the Johnson SU distribution, based on the shape parameters, scale and location. This function is an alias of PMF – JOHNSON SU.



Calculates the probability density function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \frac{\delta \exp\left(-\frac{1}{2}(\gamma + \delta \operatorname{arcsinh} z)^2\right)}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the probability density of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = PDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = PDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = PDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = PDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = PDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

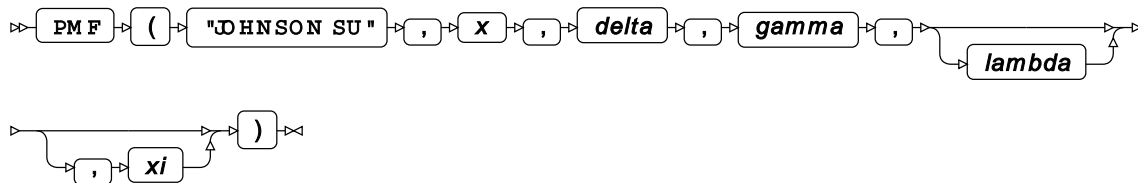
This produces the following output:

```
s1=0.3949890957
s2=0.778255654
s3=0.2647251817
s4=0.1989736025
s5=0.170842289
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

PMF – JOHNSON SU

Returns the probability mass of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of PDF – JOHNSON SU.



Calculates the probability mass function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \frac{\delta \exp\left(-\frac{1}{2}(\gamma + \delta \operatorname{arcsinh} z)^2\right)}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the probability mass of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = PMF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = PMF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = PMF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = PMF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = PMF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

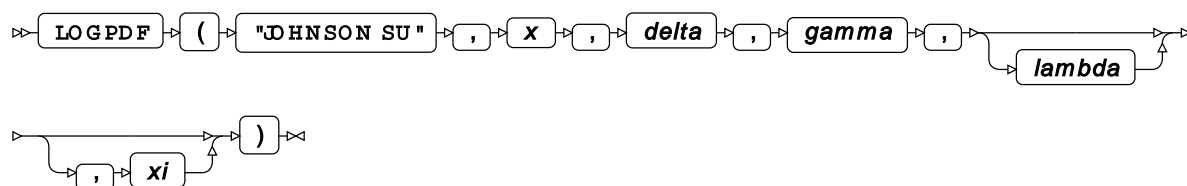
This produces the following output:

```
s1=0.3949890957
s2=0.778255654
s3=0.2647251817
s4=0.1989736025
s5=0.170842289
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGPDF – JOHNSON SU

Returns the natural logarithm of the probability density of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of LOGPMF – JOHNSON SU.



Calculates the natural logarithm of the probability density function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \frac{\delta}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} - \frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If x_i is specified, then $lambda$ must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGPDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGPDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGPDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGPDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = LOGPDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

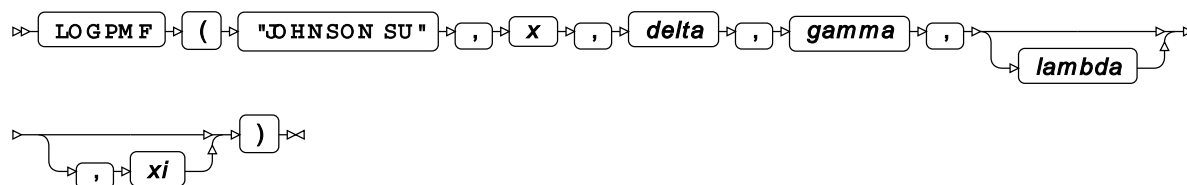
This produces the following output:

```
s1=-0.92889712
s2=-0.943847385
s3=-1.329063041
s4=-1.614583114
s5=-1.767014434
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGPMF – JOHNSON SU

Returns the natural logarithm of the probability mass of the Johnson SU distribution based on the shape parameters, scale and location. This function is an alias of LOGPDF – JOHNSON SU.



Calculates the natural logarithm of the probability mass function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \frac{\delta}{\lambda \sqrt{2\pi} \sqrt{z^2 + 1}} - \frac{1}{2} (\gamma + \delta \operatorname{arcsinh} z)^2 \quad \text{where } z = \frac{x - \xi}{\lambda}$$

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If x_i is specified, then λ must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPMF ("JOHNSON SU", 0.1, 1, 0, 1, 0);  
  PUT s1=;  
  s2 = LOGPMF ("JOHNSON SU", 0.1, 2, 0, 1, 0);  
  PUT s2=;  
  s3 = LOGPMF ("JOHNSON SU", 0.1, 1,-1, 1, 0);  
  PUT s3=;  
  s4 = LOGPMF ("JOHNSON SU", 0.1, 1, 0, 2, 0);  
  PUT s4=;  
  s5 = LOGPMF ("JOHNSON SU", 0.1, 1, 0, 1,-1);  
  PUT s5=;  
  s6 = LOGPMF ("JOHNSON SU", 0.1, 0, 0, 1,-1);  
  PUT s6=;  
RUN;
```

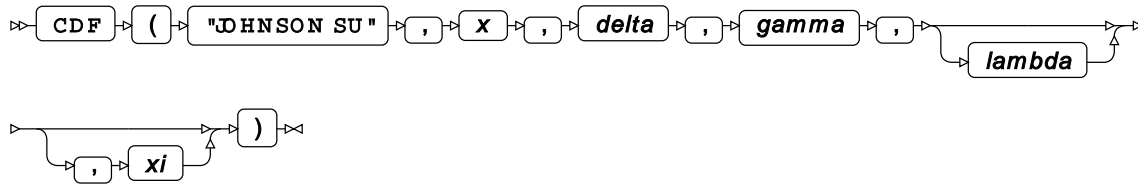
This produces the following output:

```
s1=-0.92889712  
s2=-0.943847385  
s3=-1.329063041  
s4=-1.614583114  
s5=-1.767014434  
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

CDF – JOHNSON SU

Returns the cumulative density of the Johnson SU distribution based on the shape parameters, scale and location.



Calculates the cumulative density function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \Phi\left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda}\right)$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the cumulative density of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = CDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = CDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = CDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = CDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = CDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

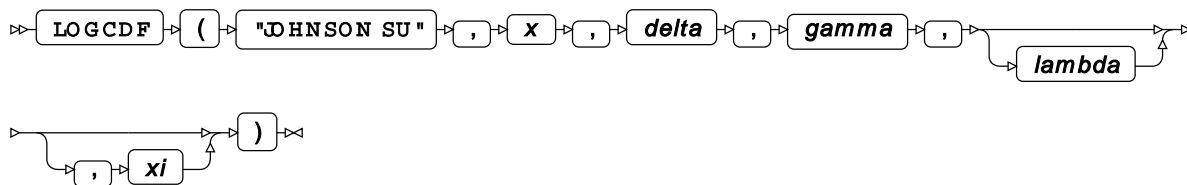
This produces the following output:

```
s1=0.5397619739
s2=0.5791299407
s3=0.1840159795
s4=0.5199305142
s5=0.8290319998
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGCDF – JOHNSON SU

Returns the natural logarithm of the cumulative density of the Johnson SU distribution based on the shape parameters, scale and location.



Calculates the natural logarithm of the cumulative density function for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right)$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGCDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGCDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGCDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGCDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = LOGCDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

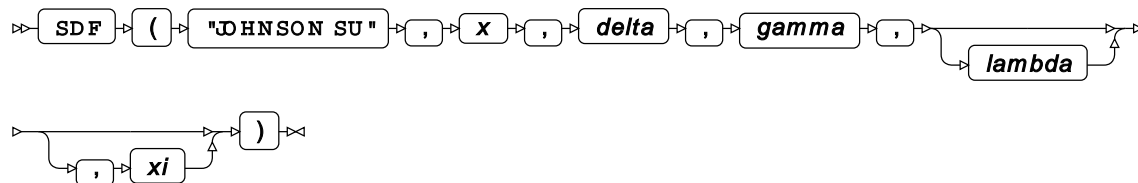
This produces the following output:

```
s1=-0.616627026
s2=-0.546228404
s3=-1.69273268
s4=-0.654060103
s5=-0.187496524
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

SDF – JOHNSON SU

Returns the survival of the Johnson SU distribution based on the shape parameters, scale and location.



Calculates the survival, or the complement to the cumulative density function, for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = 1 - \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right)$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the survival of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = SDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = SDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = SDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = SDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = SDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

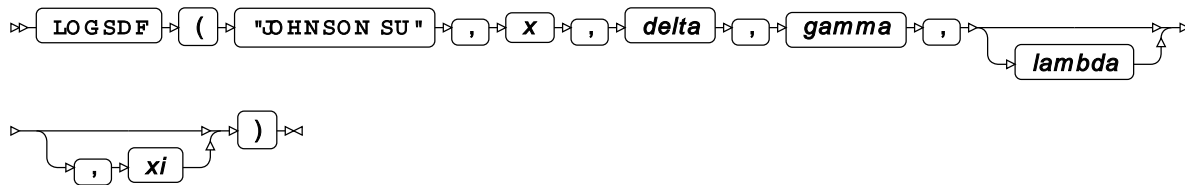
This produces the following output:

```
s1=0.4602380261
s2=0.4208700593
s3=0.8159840205
s4=0.4800694858
s5=0.1709680002
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

LOGSDF – JOHNSON SU

Returns the natural logarithm of the survival of the Johnson SU distribution based on the shape parameters, scale and location.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Johnson SU distribution at point x , based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$\delta > 0, \lambda > 0$$

$$f(x; \delta, \gamma, \lambda, \xi) = \log \left[1 - \Phi \left(\gamma + \delta \operatorname{arcsinh} \frac{x - \xi}{\lambda} \right) \right]$$

$$\Phi(x) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi(x)$ is the cumulative density function of the standard Normal distribution and $\operatorname{erf}(x)$ is the error function, see [CDF – NORMAL](#) (page 1187) and [ERF](#) (page 1830).

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = LOGSDF ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = LOGSDF ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = LOGSDF ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = LOGSDF ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = LOGSDF ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

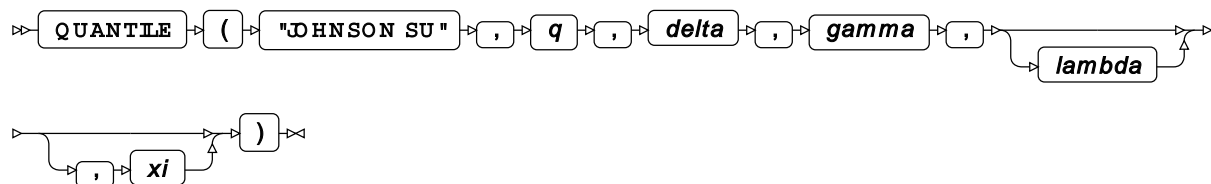
This produces the following output:

```
s1=-0.776011475
s2=-0.865431141
s3=-0.203360507
s4=-0.733824424
s5=-1.766278873
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

QUANTILE – JOHNSON SU

Returns the quantile of the Johnson SU distribution based on the shape parameters, scale and location.



Calculates the quantile x , or the inverse of the cumulative density function, for the Johnson SU distribution for probability value q based on the shape parameters δ (*delta*) and γ (*gamma*), scale λ (*lambda*) and location ξ (*xi*). Arguments *lambda* and *xi* are optional. If *lambda* is omitted, it defaults to 1; if *xi* is omitted, it defaults to 0. If *xi* is specified, then *lambda* must also be specified.

This function is defined under the following conditions:

$$0 < q < 1, \delta > 0, \lambda > 0$$

$$f(q) = \lambda \sinh \frac{\Phi^{-1}(q) - \gamma}{\delta} + \xi$$

$$\Phi^{-1}(q) = \sqrt{2} \operatorname{erf}^{-1}(2q - 1)$$

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$

where $\Phi^{-1}(q)$ is the probit function, or the quantile function of the standard Normal distribution, and $\operatorname{erf}^{-1}(q)$ is the inverse of the error function, see [PROBIT](#) (page 1200) and [ERF](#) (page 1830).

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, a missing value is returned.

gamma

Type: Numeric

The second shape parameter.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *xi* is specified, then *lambda* must also be specified.

If the argument is out of range, a missing value is returned.

xi

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *xi* is specified, then *lambda* must also be specified.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the quantile of the Johnson SU distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE ("JOHNSON SU", 0.1, 1, 0, 1, 0);
  PUT s1=;
  s2 = QUANTILE ("JOHNSON SU", 0.1, 2, 0, 1, 0);
  PUT s2=;
  s3 = QUANTILE ("JOHNSON SU", 0.1, 1,-1, 1, 0);
  PUT s3=;
  s4 = QUANTILE ("JOHNSON SU", 0.1, 1, 0, 2, 0);
  PUT s4=;
  s5 = QUANTILE ("JOHNSON SU", 0.1, 1, 0, 1,-1);
  PUT s5=;
  s6 = QUANTILE ("JOHNSON SU", 0.1, 0, 0, 1,-1);
  PUT s6=;
RUN;
```

This produces the following output:

```
s1=-1.662309119
s2=-0.685534595
s3=-0.285286163
s4=-3.324618237
s5=-2.662309119
s6=.
```

The first five examples show the impact of the arguments on the result for the same point x in the distribution. In the sixth example the function returns a missing value because one of the arguments is out of range.

RAND – JOHNSON SU

Returns a random number from the Johnson SU distribution based on the shape parameters.

```
⇒ RAND ⇒ ( ⇒ "JOHNSON SU" ⇒ , ⇒ delta ⇒ , ⇒ gamma ⇒ ) ⇒
```

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

The Johnson SU distribution is defined in: N. L. Johnson, "Systems of frequency curves generated by methods of translation", *Biometrika* 36, no. 1/2 (1949): 149-76, doi:10.2307/2332539.

Return type: Numeric

The return value is negative.

delta

Type: Numeric

The first shape parameter.

Restriction: must be positive

If the argument is out of range, an error is returned.

gamma

Type: Numeric

The second shape parameter.

Example

In this example, a random number from the Johnson SU distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("JOHNSON SU", 10,5);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-0.673348129  
-0.663680708  
-0.506524535  
-0.613047601  
-0.62250131
```

Running the DATA step again produces the following output.

```
The random numbers are:  
-0.564678432  
-0.429874148  
-0.663882461  
-0.322217153  
-0.440821654
```


However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(18);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("JOHNSON SU", 10,5);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
-0.465294973
-0.487261343
-0.556494653
-0.537014675
-0.571047881
```

Running the `DATA` step again produces the same output.

Laplace distribution

Functions for the Laplace distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – LAPLACE [↗](#)..... 1093

$$\frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

Returns the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of `PMF – LAPLACE`.

PMF – LAPLACE [↗](#)..... 1095

$$\frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

Returns the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of `PDF – LAPLACE`.

LOGPDF – LAPLACE [↗](#)..... 1097

$$\log \left[\frac{1}{2\lambda} \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right]$$

Returns the natural logarithm of the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of LOGPMF – LAPLACE.

LOGPMF – LAPLACE [↗](#)..... 1100

$$\log \left[\frac{1}{2\lambda} \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right]$$

Returns the natural logarithm of the probability mass function at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of LOGPDF – LAPLACE.

CDF – LAPLACE [↗](#)..... 1102

$$\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(x-\theta) \left(1 - \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right)$$

Returns the cumulative density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.

LOGCDF – LAPLACE [↗](#)..... 1105

$$\log \left[\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(x-\theta) \left(1 - \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right) \right]$$

Returns the natural logarithm of the cumulative density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.

SDF – LAPLACE [↗](#)..... 1107

$$\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\theta-x) \left(1 - \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right)$$

Returns the survival density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.

LOGSDF – LAPLACE [↗](#)..... 1110

$$\log \left[\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\theta-x) \left(1 - \exp \left(-\frac{|x-\theta|}{\lambda} \right) \right) \right]$$

Returns the natural logarithm of the survival density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.

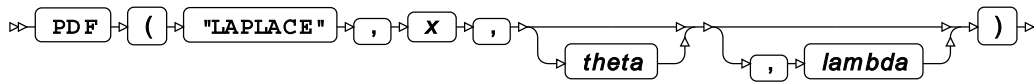
QUANTILE – LAPLACE [↗](#)..... 1112

$$\theta - \lambda \cdot \operatorname{sgn}(q-0.5) \cdot \log(1-2|q-0.5|)$$

Returns the quantile of a Laplace distribution (also known as the Double Exponential distribution) for a specified cumulative density, with specified location and scale parameters.

PDF – LAPLACE

Returns the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of PMF – LAPLACE.



The probability density function at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a 'standard' Laplace distribution is assumed, with $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The probability density function at a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined by two exponential distributions as follows:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \begin{cases} \exp\left(-\frac{\theta-x}{\lambda}\right) & \text{if } x < \theta \\ \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

θ

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

λ

Optional argument

Type: Numeric

The scale parameter. If used, θ must also be specified.

Restriction: $\lambda > 0$.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= PDF ("LAPLACE", 1, 0, 1);  
PUT s1=;  
s2= PDF ("LAPLACE", 1, 0);  
PUT s2=;  
s3= PDF ("LAPLACE", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=0.1839397206  
s2=0.1839397206  
s3=0.1839397206
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = PDF ("LAPLACE", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.0705401437
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a much lower probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = PDF ("LAPLACE", 5, 0, 1);  
PUT s5=;  
s6 = PDF ("LAPLACE", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=0.0033689735  
s6=0.0033689735
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = PDF("LAPLACE", 1, 0, 1, 0);
PUT s7=;
s8 = PDF("LAPLACE", 1, , 0);
PUT s8=;
RUN;
```

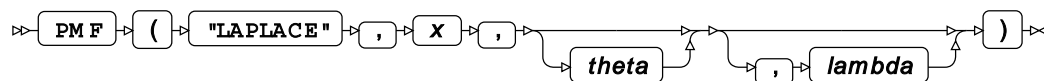
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

PMF – LAPLACE

Returns the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of PDF – LAPLACE.



The probability density function at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a 'standard' Laplace distribution is assumed, with $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The probability density function at a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined by two exponential distributions as follows:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \begin{cases} \exp\left(-\frac{\theta-x}{\lambda}\right) & \text{if } x < \theta \\ \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= PMF("LAPLACE", 1, 0, 1);  
PUT s1=;  
s2= PMF("LAPLACE", 1, 0);  
PUT s2=;  
s3= PMF("LAPLACE", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=0.1839397206  
s2=0.1839397206  
s3=0.1839397206
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = PMF("LAPLACE", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=0.0705401437
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a much lower probability density value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
s5 = PMF("LAPLACE", 5, 0, 1);
PUT s5=;
s6 = PMF("LAPLACE", 2, -3, 1);
PUT s6=;
RUN;
```

This produces the following output:

```
s5=0.0033689735
s6=0.0033689735
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = PMF("LAPLACE", 1, 0, 1, 0);
PUT s7=;
s8 = PMF("LAPLACE", 1, , 0);
PUT s8=;
RUN;
```

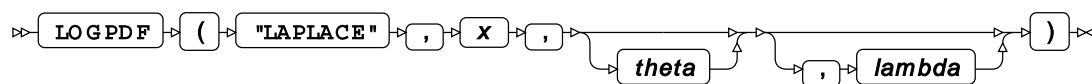
This produces the following output:

```
s7= .
s8= .
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

LOGPDF – LAPLACE

Returns the natural logarithm of the probability density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of LOGPMF – LAPLACE.



The probability density function (PDF) at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, with $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The probability density function at a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined by two exponential distributions as follows:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \begin{cases} \exp\left(-\frac{\theta-x}{\lambda}\right) & \text{if } x < \theta \\ \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

The natural logarithm of this is therefore:

$$f(x; \theta, \lambda) = \log\left[\frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)\right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGPDF("LAPLACE", 1, 0, 1);  
PUT s1=;  
s2= LOGPDF("LAPLACE", 1, 0);  
PUT s2=;  
s3= LOGPDF("LAPLACE", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-1.693147181  
s2=-1.693147181  
s3=-1.693147181
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGPDF("LAPLACE", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-2.651573316
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower natural logarithm of probability density at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
s5 = LOGPDF("LAPLACE", 5, 0, 1);  
PUT s5=;  
s6 = LOGPDF("LAPLACE", 2, -3, 1);  
PUT s6=;  
RUN;
```

This produces the following output:

```
s5=-5.693147181  
s6=-5.693147181
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = LOGPDF("LAPLACE", 1, 0, 1, 0);
PUT s7=;
s8 = LOGPDF("LAPLACE", 1, , 0);
PUT s8=;
RUN;
```

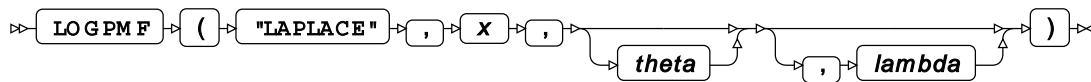
This produces the following output:

```
s7=.
s8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

LOGPMF – LAPLACE

Returns the natural logarithm of the probability mass function at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters. This function is an alias of LOGPDF – LAPLACE.



The probability mass function (PMF) at a point x gives the likelihood that a randomly drawn value from the distribution is equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, with $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The probability mass function function at a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined as:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \begin{cases} \exp\left(-\frac{\theta-x}{\lambda}\right) & \text{if } x < \theta \\ \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)$$

The natural logarithm of this is therefore:

$$f(x; \theta, \lambda) = \log\left[\frac{1}{2\lambda} \exp\left(-\frac{|x-\theta|}{\lambda}\right)\right]$$

Return type: Numeric

x**Type:** Numeric

The point at which to calculate the natural logarithm of the probability mass function.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
s1= LOGPMF("LAPLACE", 1, 0, 1);  
PUT s1=;  
s2= LOGPMF("LAPLACE", 1, 0);  
PUT s2=;  
s3= LOGPMF("LAPLACE", 1);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-1.693147181  
s2=-1.693147181  
s3=-1.693147181
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
s4 = LOGPMF("LAPLACE", 1, 0, 6);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s4=-2.651573316
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower natural logarithm of probability density at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
s5 = LOGPMF("LAPLACE", 5, 0, 1);
PUT s5=;
s6 = LOGPMF("LAPLACE", 2, -3, 1);
PUT s6=;
RUN;
```

This produces the following output:

```
s5=-5.693147181
s6=-5.693147181
```

These two examples demonstrate the location parameter. The first example evaluates the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same relative point on the distribution, but with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
s7 = LOGPMF("LAPLACE", 1, 0, 1, 0);
PUT s7=;
s8 = LOGPMF("LAPLACE", 1, , 0);
PUT s8=;
RUN;
```

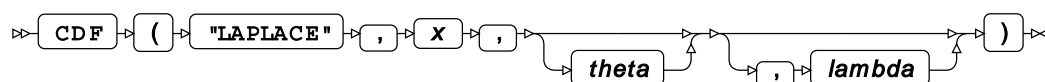
This produces the following output:

```
s7=.
s8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

CDF – LAPLACE

Returns the cumulative density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.



The cumulative density function (CDF) at a point x gives the likelihood that a randomly drawn value from the distribution is less than or equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, which has $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The cumulative density function for a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined as the integral of the PDF from minus infinity to x :

$$f(x; \theta, \lambda) = \int_{-\infty}^x f(u) du = \begin{cases} \frac{1}{2} \exp\left(\frac{x-\theta}{\lambda}\right) & \text{if } x < \theta \\ 1 - \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2} + \frac{1}{2} \operatorname{sgn}(x - \theta) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right)\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1= CDF ("LAPLACE", 1, 0, 1);
PUT S1=;
S2= CDF ("LAPLACE", 1, 0);
PUT S2=;
S3= CDF ("LAPLACE", 1);
PUT S3=;
RUN;
```

This produces the following output:

```
S1=0.8160602794  
S2=0.8160602794  
S3=0.8160602794
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = CDF("LAPLACE", 1, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=0.5767591376
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
S5 = CDF("LAPLACE", 5, 0, 1);  
PUT S5=;  
S6 = CDF("LAPLACE", 2, -3, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.9966310265  
S6=0.9966310265
```

These two examples demonstrate the location parameter. The first example gives the cumulative density of the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;  
S7 = CDF("LAPLACE", 1, 0, 1, 0);  
PUT S7=;  
S8 = CDF("LAPLACE", 1, , 0);  
PUT S8=;  
RUN;
```

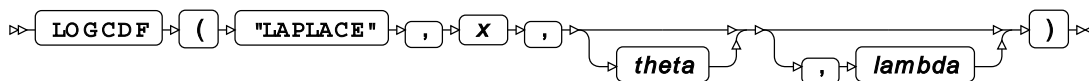
This produces the following output:

```
S7=.
S8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

LOGCDF – LAPLACE

Returns the natural logarithm of the cumulative density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.



The cumulative density function (CDF) at a point x gives the likelihood that a randomly drawn value from the distribution is less than or equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, which has $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The cumulative density function for a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined as the integral of the PDF from minus infinity to x :

$$f(x; \theta, \lambda) = \int_{-\infty}^x f(u) du = \begin{cases} \frac{1}{2} \exp\left(\frac{x-\theta}{\lambda}\right) & \text{if } x < \theta \\ 1 - \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2} + \frac{1}{2} \operatorname{sgn}(x - \theta) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right) \right)$$

The natural logarithm of this is then defined as:

$$f(x; \theta, \lambda) = \log \left[\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(x - \theta) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right) \right) \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1= LOGCDF("LAPLACE", 1, 0, 1);  
PUT S1=;  
S2= LOGCDF("LAPLACE", 1, 0);  
PUT S2=;  
S3= LOGCDF("LAPLACE", 1);  
PUT S3=;  
RUN;
```

This produces the following output:

```
S1=-0.203267055  
S2=-0.203267055  
S3=-0.203267055
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a 'standard' Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = LOGCDF("LAPLACE", 1, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=-0.550330539
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a lower value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
S5 = LOGCDF("LAPLACE", 5, 0, 1);
PUT S5=;
S6 = LOGCDF("LAPLACE", 2, -3, 1);
PUT S6=;
RUN;
```

This produces the following output:

```
S5=-0.003374661
S6=-0.003374661
```

These two examples demonstrate the location parameter. The first example gives the natural logarithm of the cumulative density of the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
S7 = LOGCDF("LAPLACE", 1, 0, 1, 0);
PUT S7=;
S8 = LOGCDF("LAPLACE", 1, , 0);
PUT S8=;
RUN;
```

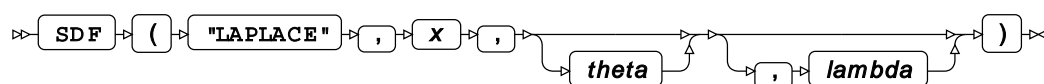
This produces the following output:

```
S7=.
S8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

SDF – LAPLACE

Returns the survival density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.



The survival density function (SDF) at a point x gives the likelihood that a randomly drawn value from the distribution is greater than or equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, which has $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The survival function for a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined as the integral of the PDF from x to infinity:

$$f(x; \theta, \lambda) = \int_x^{\infty} f(u) du = \begin{cases} 1 - \frac{1}{2} \exp\left(\frac{x-\theta}{\lambda}\right) & \text{if } x < \theta \\ \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\theta - x) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right)\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival density.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1= SDF("LAPLACE", 1, 0, 1);
PUT S1=;
S2= SDF("LAPLACE", 1, 0);
PUT S2=;
S3= SDF("LAPLACE", 1);
PUT S3=;
RUN;
```

This produces the following output:

```
S1=0.1839397206  
S2=0.1839397206  
S3=0.1839397206
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a standard Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = SDF("LAPLACE", 1, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=0.4232408624
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a larger value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
S5 = SDF("LAPLACE", 5, 0, 1);  
PUT S5=;  
S6 = SDF("LAPLACE", 2, -3, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.0033689735  
S6=0.0033689735  
S7=.  
S8=.
```

These two examples demonstrate the location parameter. The first example gives the value of the SDF of the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;  
S7 = SDF("LAPLACE", 1, 0, 1, 0);  
PUT S7=;  
S8 = SDF("LAPLACE", 1, , 0);  
PUT S8=;  
RUN;
```

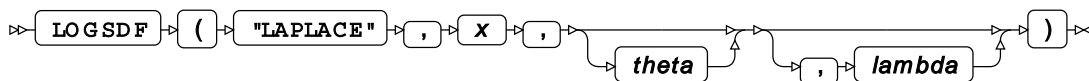
This produces the following output:

```
S7= .
S8= .
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

LOGSDF – LAPLACE

Returns the natural logarithm of the survival density at a point from a Laplace distribution (also known as the Double Exponential distribution), with specified location and scale parameters.



The survival density function (SDF) at a point x gives the likelihood that a randomly drawn value from the distribution is greater than or equal to x .

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a standard Laplace distribution is assumed, which has $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The survival function for a point x of a Laplace distribution with location parameter θ and scale parameter λ is defined as the integral of the PDF from x to infinity:

$$f(x; \theta, \lambda) = \int_x^{\infty} f(u) du = \begin{cases} 1 - \frac{1}{2} \exp\left(\frac{x-\theta}{\lambda}\right) & \text{if } x < \theta \\ \frac{1}{2} \exp\left(-\frac{x-\theta}{\lambda}\right) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(x; \theta, \lambda) = \frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\theta - x) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right) \right)$$

The natural logarithm of this is defined as:

$$f(x; \theta, \lambda) = \log \left[\frac{1}{2} + \frac{1}{2} \operatorname{sgn}(\theta - x) \left(1 - \exp\left(-\frac{|x - \theta|}{\lambda}\right) \right) \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival density .

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: *lambda* > 0.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1= LOGSDF("LAPLACE", 1, 0, 1);  
PUT S1=;  
S2= LOGSDF("LAPLACE", 1, 0);  
PUT S2=;  
S3= LOGSDF("LAPLACE", 1);  
PUT S3=;  
RUN;
```

This produces the following output:

```
S1=-1.693147181  
S2=-1.693147181  
S3=-1.693147181
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each resolving to $x = 1$, $\theta = 0$, and $\lambda = 1$ (a standard Laplace distribution) and therefore returning the same value.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = LOGSDF("LAPLACE", 1, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=-0.859813847
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This returns a larger value at $x = 1$ than the basic examples.

Example – use of location parameter

```
DATA _NULL_;
S5 = LOGSDF("LAPLACE", 5, 0, 1);
PUT S5=;
S6 = LOGSDF("LAPLACE", 2, -3, 1);
PUT S6=;
RUN;
```

This produces the following output:

```
S5=-5.693147181
S6=-5.693147181
```

These two examples demonstrate the location parameter. The first example gives the natural logarithm of the value of the SDF of the distribution at $x = 5$, with $\theta = 0$ and $\lambda = 1$; whereas the second shifts the distribution back using $\theta = -3$, but also shifts x back by the same amount, to $x = 2$. Both produce the same result, because they evaluate the function at the same point of the distribution, just with the distribution located in different places.

Example – invalid syntax

```
DATA _NULL_;
S7 = LOGSDF("LAPLACE", 1, 0, 1, 0);
PUT S7=;
S8 = LOGSDF("LAPLACE", 1, , 0);
PUT S8=;
RUN;
```

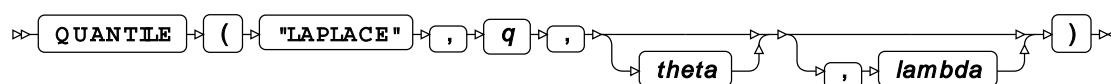
This produces the following output:

```
S7=.
S8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

QUANTILE – LAPLACE

Returns the quantile of a Laplace distribution (also known as the Double Exponential distribution) for a specified cumulative density, with specified location and scale parameters.



The quantile for a cumulative density q gives a threshold value of x , below which randomly drawn values from the defined Laplace distribution should occur q times.

You can optionally specify the distribution's location parameter θ (theta) and scale parameter λ (lambda). If these arguments are not specified, a 'standard' Laplace distribution is assumed, which has $\theta = 0$ and $\lambda = 1$. It is possible to specify θ (theta) without λ (lambda) (in which case, λ will be set to 1), but not λ (lambda) without θ (theta).

The quantile function for a Laplace distribution with location parameter θ and scale parameter λ is defined as the inverse of the cumulative density function for a Laplace distribution:

$$f(q; \lambda, \theta) = \begin{cases} \theta - \lambda(\log(2 - 2q)) & \text{if } x < \theta \\ \theta + \lambda(\log(2q)) & \text{if } x \geq \theta \end{cases}$$

which simplifies to:

$$f(q; \lambda, \theta) = \theta - \lambda \cdot \text{sgn}(q - 0.5) \cdot \log(1 - 2|q - 0.5|)$$

Return type: Numeric

q

Type: Numeric

The cumulative probability value for which to calculate the quantile.

Restriction: $0 < q < 1$

If the argument is out of range, a missing value is returned.

theta

Optional argument

Type: Numeric

The location parameter. This defines the location of the distribution's centre of symmetry on the x axis.

lambda

Optional argument

Type: Numeric

The scale parameter. If used, *theta* must also be specified.

Restriction: $\lambda \geq 0$.

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1= QUANTILE("LAPLACE", 0.81606027942, 0, 1);  
PUT S1=;  
S2= QUANTILE("LAPLACE", 0.81606027942, 0);  
PUT S2=;  
S3= QUANTILE("LAPLACE", 0.81606027942);  
PUT S3=;  
RUN;
```

This produces the following output:

```
S1=1  
S2=1  
S3=1
```

These first three examples are all equivalent and demonstrate the flexible syntax, with each specifying $q = 0.81606027942$, and in every case resolving to $\theta = 0$, and $\lambda = 1$; therefore returning the same value: $x = 1$.

Example – use of scale parameter

```
DATA _NULL_;  
S4 = QUANTILE("LAPLACE", 0.69220062755, 0, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S4=1
```

This example demonstrates the scale parameter, by retaining the basic examples' $x = 1$ and $\theta = 0$, but setting $\lambda = 6$ to reduce the height and increase the dispersion of the distribution. This means that a smaller cumulative density is required to return an x value of 1 compared to the basic examples.

Example – use of location parameter

```
DATA _NULL_;  
S5 = QUANTILE("LAPLACE", 0.9966310265005, 0, 1);  
PUT S5=;  
S6 = QUANTILE("LAPLACE", 0.9966310265005, -3, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=5  
S6=2
```

These two examples demonstrate the location parameter. The same cumulative density is input, but with the Laplace distribution at two different locations, thus producing different values of x .

Example – invalid syntax

```
DATA _NULL_;
S7 = QUANTILE("LAPLACE", 0.9966310265005, 1, 0, 1, 0);
PUT S7=;
S8 = QUANTILE("LAPLACE", 0.9966310265005, 1, , 0);
PUT S8=;
RUN;
```

This produces the following output:

```
S7=.
S8=.
```

Neither of these examples are valid; the first has too many parameters, and the second specifies λ without θ .

Logistic distribution

Functions for the Logistic distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – LOGISTIC [↗](#)..... 1116

$$\frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)$$

Returns the probability density of the Logistic distribution based on the location and scale parameters. This function is an alias of PMF – LOGISTIC.

PMF – LOGISTIC [↗](#)..... 1118

$$\frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)$$

Returns the probability mass of the Logistic distribution based on the location and scale parameters. This function is an alias of PDF – LOGISTIC.

LOGPDF – LOGISTIC [↗](#)..... 1119

$$\log \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \log\left[\frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)\right]$$

Returns the natural logarithm of the probability density of the Logistic distribution based on the location and scale parameters. This function is an alias of LOGPMF – LOGISTIC.

LOGPMF – LOGISTIC [↗](#)..... 1121

$$\log \frac{\exp \left(-\frac{x-\mu}{\sigma} \right)}{\sigma \left(1 + \exp \left(-\frac{x-\mu}{\sigma} \right) \right)^2} = \log \left[\frac{1}{4\sigma} \operatorname{sech}^2 \left(\frac{x-\mu}{2\sigma} \right) \right]$$

Returns the natural logarithm of the probability mass of the Logistic distribution based on the location and scale parameters. This function is an alias of LOGPDF – LOGISTIC.

CDF – LOGISTIC [↗](#)..... 1123

$$\frac{1}{1 + \exp \left(-\frac{x-\mu}{\sigma} \right)} = \frac{1}{2} + \frac{1}{2} \tanh \frac{x-\mu}{2\sigma}$$

Returns the cumulative density of the Logistic distribution based on the location and scale parameters.

LOGCDF – LOGISTIC [↗](#)..... 1124

$$\log \frac{1}{1 + \exp \left(-\frac{x-\mu}{\sigma} \right)} = \log \left[\frac{1}{2} + \frac{1}{2} \tanh \frac{x-\mu}{2\sigma} \right]$$

Returns the natural logarithm of the cumulative density of the Logistic distribution based on the location and scale parameters.

SDF – LOGISTIC [↗](#)..... 1126

$$\frac{1}{1 + \exp \left(\frac{x-\mu}{\sigma} \right)} = \frac{1}{2} - \frac{1}{2} \tanh \frac{x-\mu}{2\sigma}$$

Returns the survival of the Logistic distribution based on the location and scale parameters.

LOGSDF – LOGISTIC [↗](#)..... 1127

$$\log \frac{1}{1 + \exp \left(\frac{x-\mu}{\sigma} \right)} = \log \left[\frac{1}{2} - \frac{1}{2} \tanh \frac{x-\mu}{2\sigma} \right]$$

Returns the natural logarithm of the survival of the Logistic distribution based on the location and scale parameters.

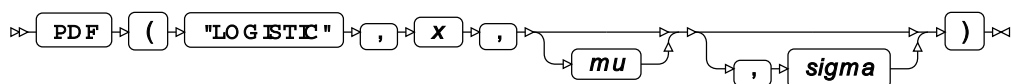
QUANTILE – LOGISTIC [↗](#)..... 1129

$$f(q) = \mu + \sigma \log \frac{q}{1-q}$$

Returns the quantile of the Logistic distribution for a specified probability value based on the location and scale parameters.

PDF – LOGISTIC

Returns the probability density of the Logistic distribution based on the location and scale parameters. This function is an alias of PMF – LOGISTIC.



Calculates the probability density function for the Logistic distribution at point x , based on the location parameter μ (*mu*) and the scale parameter σ (*sigma*).

If *sigma* is specified, then *mu* must also be specified.

$$f(x; \mu, \sigma) = \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1 + \exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("LOGISTIC", 0.5, -1, 4);
  PUT s1=;
  s2 = PDF ("LOGISTIC", 0.5, 0, 1);
  PUT s2=;
  s3 = PDF ("LOGISTIC", -0.5, 0, 1);
  PUT s3=;
  s4 = PDF ("LOGISTIC", 0.5, -1, -5);
  PUT s4=;
RUN;
```

This produces the following output:

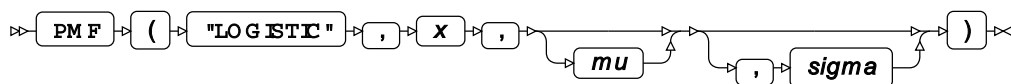
```
s1=0.0603532253
s2=0.2350037122
s3=0.2350037122
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The probability density function of the Logistic distribution is symmetrical around the location parameter μ as illustrated in the second and third examples. The function returns the same result for points which are equidistant from μ .

PMF – LOGISTIC

Returns the probability mass of the Logistic distribution based on the location and scale parameters. This function is an alias of PDF – LOGISTIC.



Calculates the probability mass function for the Logistic distribution at point x , based on the location parameter μ (mu) and the scale parameter σ ($sigma$).

If $sigma$ is specified, then mu must also be specified.

$$f(x; \mu, \sigma) = \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma(1 + \exp\left(-\frac{x-\mu}{\sigma}\right))^2} = \frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If $sigma$ is specified, then mu must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1**Restriction:** must be positiveIf *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PMF ("LOGISTIC", 0.5, -1, 4);
  PUT s1=;
  s2 = PMF ("LOGISTIC", 0.5, 0, 1);
  PUT s2=;
  s3 = PMF ("LOGISTIC", -0.5, 0, 1);
  PUT s3=;
  s4 = PMF ("LOGISTIC", 0.5, -1, -5);
  PUT s4=;
RUN;
```

This produces the following output:

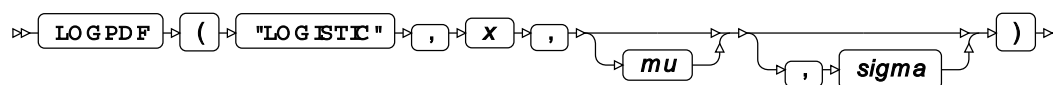
```
s1=0.0603532253
s2=0.2350037122
s3=0.2350037122
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The probability mass function of the Logistic distribution is symmetrical around the location parameter μ as illustrated in the second and third examples. The function returns the same result for points which are equidistant from μ .

LOGPDF – LOGISTIC

Returns the natural logarithm of the probability density of the Logistic distribution based on the location and scale parameters. This function is an alias of LOGPMF – LOGISTIC.



Calculates the natural logarithm of the probability density function for the Logistic distribution at point x , based on the location parameter μ (mu) and the scale parameter σ ($sigma$).

If $sigma$ is specified, then mu must also be specified.

$$f(x; \mu, \sigma) = \log \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1 + \exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \log\left[\frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right)\right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If $sigma$ is specified, then mu must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

$sigma$

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If $sigma$ is specified, then mu must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("LOGISTIC", 0.5, -1, 4);
  PUT s1=;
  s2 = LOGPDF ("LOGISTIC", 0.5, 0, 1);
  PUT s2=;
  s3 = LOGPDF ("LOGISTIC", -0.5, 0, 1);
  PUT s3=;
  s4 = LOGPDF ("LOGISTIC", 0.5, -1, -5);
  PUT s4=;
RUN;
```

This produces the following output:

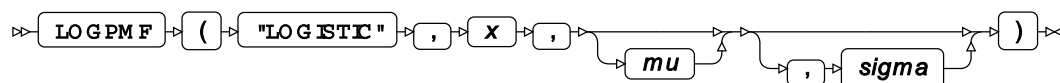
```
s1=-2.807540889
s2=-1.448153968
s3=-1.448153968
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The natural logarithm of the probability density function of the Logistic distribution is symmetrical around the location parameter μ as illustrated in the second and third examples. The function returns the same result for points which are equidistant from μ .

LOGPMF – LOGISTIC

Returns the natural logarithm of the probability mass of the Logistic distribution based on the location and scale parameters. This function is an alias of LOGPDF – LOGISTIC.



Calculates the natural logarithm of the probability mass function for the Logistic distribution at point x , based on the location parameter μ (mu) and the scale parameter σ ($sigma$).

If $sigma$ is specified, then mu must also be specified.

$$f(x; \mu, \sigma) = \log \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma \left(1 + \exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2} = \log \left[\frac{1}{4\sigma} \operatorname{sech}^2\left(\frac{x-\mu}{2\sigma}\right) \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPMF ("LOGISTIC", 0.5, -1, 4);  
  PUT s1=;  
  s2 = LOGPMF ("LOGISTIC", 0.5, 0, 1);  
  PUT s2=;  
  s3 = LOGPMF ("LOGISTIC", -0.5, 0, 1);  
  PUT s3=;  
  s4 = LOGPMF ("LOGISTIC", 0.5, -1, -5);  
  PUT s4=;  
RUN;
```

This produces the following output:

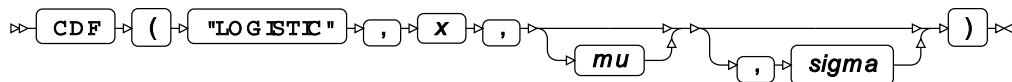
```
s1=-2.807540889  
s2=-1.448153968  
s3=-1.448153968  
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

The natural logarithm of the probability mass function of the Logistic distribution is symmetrical around the location parameter μ as illustrated in the second and third examples. The function returns the same result for points which are equidistant from μ .

CDF – LOGISTIC

Returns the cumulative density of the Logistic distribution based on the location and scale parameters.



Calculates the cumulative density function for the Logistic distribution at point x , based on the location parameter μ (*mu*) and the scale parameter σ (*sigma*).

If *sigma* is specified, then *mu* must also be specified.

$$f(x; \mu, \sigma) = \frac{1}{1 + \exp\left(-\frac{x-\mu}{\sigma}\right)} = \frac{1}{2} + \frac{1}{2} \tanh \frac{x-\mu}{2\sigma}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF ("LOGISTIC", -1, -1, 4);
  PUT s1=;
  s2 = CDF ("LOGISTIC", 0.5, 0, 1);
  PUT s2=;
  s3 = CDF ("LOGISTIC", -0.5, 0, 1);
  PUT s3=;
  s4 = CDF ("LOGISTIC", 0.5, -1, -5);
  PUT s4=;
RUN;
```

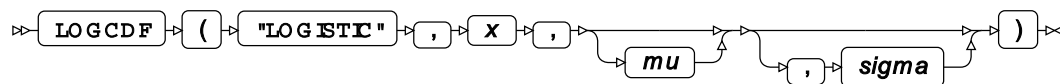
This produces the following output:

```
s1=0.5
s2=0.6224593312
s3=0.3775406688
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

LOGCDF – LOGISTIC

Returns the natural logarithm of the cumulative density of the Logistic distribution based on the location and scale parameters.



Calculates the natural logarithm of the cumulative density function for the Logistic distribution at point x , based on the location parameter μ (mu) and the scale parameter σ ($sigma$).

If $sigma$ is specified, then mu must also be specified.

$$f(x; \mu, \sigma) = \log \frac{1}{1 + \exp\left(-\frac{x-\mu}{\sigma}\right)} = \log \left[\frac{1}{2} + \frac{1}{2} \tanh \frac{x-\mu}{2\sigma} \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

mu

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGCDF ("LOGISTIC",-1,-1,4);  
  PUT s1=;  
  s2 = LOGCDF ("LOGISTIC",0.5,0,1);  
  PUT s2=;  
  s3 = LOGCDF ("LOGISTIC",-0.5,0,1);  
  PUT s3=;  
  s4 = LOGCDF ("LOGISTIC",0.5,-1,-5);  
  PUT s4=;  
RUN;
```

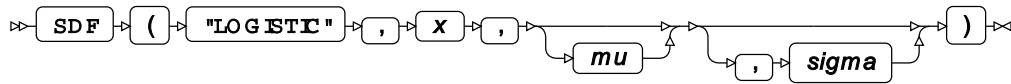
This produces the following output:

```
s1=-0.693147181  
s2=-0.474076984  
s3=-0.974076984  
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

SDF – LOGISTIC

Returns the survival of the Logistic distribution based on the location and scale parameters.



Calculates the survival, or the complement to the cumulative density function, for the Logistic distribution at point x , based on the location parameter μ (*mu*) and the scale parameter σ (*sigma*).

$$f(x; \mu, \sigma) = \frac{1}{1 + \exp\left(\frac{x - \mu}{\sigma}\right)} = \frac{1}{2} - \frac{1}{2} \tanh \frac{x - \mu}{2\sigma}$$

Return type: Numeric

X

Type: Numeric

The point at which to calculate the survival.

 μ

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

sigma

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF ("LOGISTIC", -1, -1, 4);
  PUT s1=;
  s2 = SDF ("LOGISTIC", 0.5, 0, 1);
  PUT s2=;
  s3 = SDF ("LOGISTIC", -0.5, 0, 1);
  PUT s3=;
  s4 = SDF ("LOGISTIC", 0.5, -1, -5);
  PUT s4=;
RUN;
```

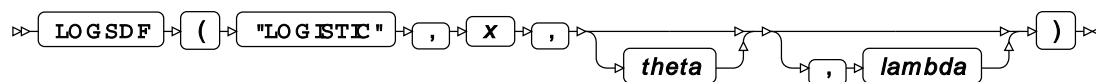
This produces the following output:

```
s1=0.5
s2=0.3775406688
s3=0.6224593312
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

LOGSDF – LOGISTIC

Returns the natural logarithm of the survival of the Logistic distribution based on the location and scale parameters.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Logistic distribution at point x , based on the location parameter μ (mu) and the scale parameter σ ($sigma$).

If $sigma$ is specified, then mu must also be specified.

$$f(x; \mu, \sigma) = \log \frac{1}{1 + \exp\left(\frac{x-\mu}{\sigma}\right)} = \log \left[\frac{1}{2} - \frac{1}{2} \tanh \frac{x-\mu}{2\sigma} \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

$theta$

Optional argument

Type: Numeric

The location parameter.

Default: 0

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

lambda

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If *sigma* is specified, then *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGSDF ("LOGISTIC", -1, -1, 4);  
  PUT s1=;  
  s2 = LOGSDF ("LOGISTIC", 0.5, 0, 1);  
  PUT s2=;  
  s3 = LOGSDF ("LOGISTIC", -0.5, 0, 1);  
  PUT s3=;  
  s4 = LOGSDF ("LOGISTIC", 0.5, -1, -5);  
  PUT s4=;  
RUN;
```

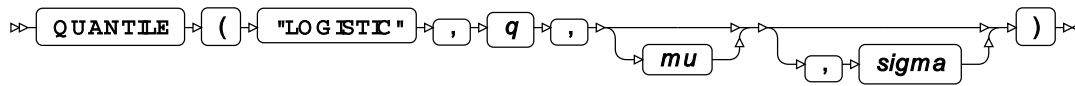
This produces the following output:

```
s1=-0.693147181  
s2=-0.974076984  
s3=-0.474076984  
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

QUANTILE – LOGISTIC

Returns the quantile of the Logistic distribution for a specified probability value based on the location and scale parameters.



Calculates the quantile x , or the inverse of the cumulative density function, for the Logistic distribution for probability value q . The quantile function returns point x such that randomly drawn values from the distribution fall below x with probability q .

If σ is specified, then μ must also be specified.

$$f(q) = \mu + \sigma \log \frac{q}{1-q}$$

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

μ

Optional argument

Type: Numeric

The location parameter.

Default: 0

If σ is specified, then μ must also be specified; otherwise a missing value is returned.

If the argument contains a missing value, a missing value is returned.

σ

Optional argument

Type: Numeric

The scale parameter.

Default: 1

Restriction: must be positive

If σ is specified, then μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Logistic distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE ("LOGISTIC", 0.1, -1, 4);
  PUT s1=;
  s2 = QUANTILE ("LOGISTIC", 0.9, -1, 4);
  PUT s2=;
  s3 = QUANTILE ("LOGISTIC", 0.5, -1, 4);
  PUT s3=;
  s4 = QUANTILE ("LOGISTIC", -0.5, -1, 4);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=-9.788898309
s2=7.7888983093
s3=-1
s4=.
```

The first three examples show the output when x lies within the domain bounds. The fourth example shows the output when x falls outside the domain bounds.

Lognormal distribution

Functions for the Lognormal distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – LOGNORMAL [↗](#)..... 1132

$$\frac{1}{x\sigma\sqrt{2\pi}} \exp\left(\frac{-(\log(x)-\mu)^2}{2\sigma^2}\right)$$

Returns the value of the probability density function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of PMF – LOGNORMAL.

PMF – LOGNORMAL [↗](#)..... 1134

$$\frac{1}{x\sigma\sqrt{2\pi}} \exp\left(\frac{-(\log(x)-\mu)^2}{2\sigma^2}\right)$$

Returns the value of the probability mass function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of PDF – LOGNORMAL.

LOGPDF – LOGNORMAL [↗](#)..... 1137

$$\log \left[\frac{1}{x\sigma\sqrt{2\pi}} \exp \left(-\frac{(\log(x) - \mu)^2}{2\sigma^2} \right) \right]$$

Returns the value of the natural logarithm of the probability density function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of LOGPMF – LOGNORMAL.

LOGPMF – LOGNORMAL [↗](#)..... 1139

$$\log \left[\frac{1}{x\sigma\sqrt{2\pi}} \exp \left(-\frac{(\log(x) - \mu)^2}{2\sigma^2} \right) \right]$$

Returns the value of the natural logarithm of the probability mass function at a specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – LOGNORMAL.

CDF – LOGNORMAL [↗](#)..... 1142

$$\frac{1}{2} + \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2}\sigma} \right]$$

Returns the value of the cumulative density function at the specified point for the Lognormal distribution with the specified mean and standard deviation.

LOGCDF – LOGNORMAL [↗](#)..... 1144

$$\log \left[\frac{1}{2} + \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2}\sigma} \right] \right]$$

Returns the value of the natural logarithm of the cumulative density function at the specified point for the Lognormal distribution with the specified mean and standard deviation.

SDF – LOGNORMAL [↗](#)..... 1147

$$\frac{1}{2} - \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2}\sigma} \right]$$

Returns the value of the survival function at the specified point for the Lognormal distribution with the specified mean and standard deviation.

LOGSDF – LOGNORMAL [↗](#)..... 1149

$$\log \left[\frac{1}{2} - \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2}\sigma} \right] \right]$$

Returns the value of the natural logarithm of the survival function at the specified point for the Lognormal distribution with the specified mean and standard deviation.

QUANTILE – LOGNORMAL [↗](#)..... 1152

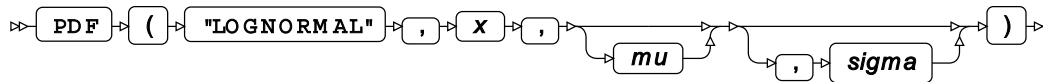
$$\inf \{x : q \leq \operatorname{CDF}(x; \mu, \sigma)\}$$

Returns the value of the quantile function at the specified point for the Lognormal distribution with the specified mean and standard deviation.

RAND – LOGNORMAL [↗](#).....1154
Returns a random number from the Lognormal distribution.

PDF – LOGNORMAL

Returns the value of the probability density function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of PMF – LOGNORMAL.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu)^2}{2\sigma^2}\right)$$

where μ is the mean of the natural logarithm of the random variable and σ is the standard deviation of the natural logarithm of the random variable.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = PDF("LOGNORMAL", 1.5, 0, 1);
  PUT s1=;
  s2 = PDF("LOGNORMAL", 1.5, 0);
  PUT s2=;
  s3 = PDF("LOGNORMAL", 1.5);
  PUT s3=;

  s4 = PDF("LOGNORMAL", 3, 2.5, 1.2);
  PUT s4=;
  s5 = PDF("LOGNORMAL", 12, 2.5, 1.2);
  PUT s5=;
  s6 = PDF("LOGNORMAL", 40, 2.5, 1.2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.2449736517
s2=0.2449736517
s3=0.2449736517
s4=0.056035057
s5=0.0277021337
s6=0.0050877715
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, PDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("LOGNORMAL", 1.5, 1, 2, 3);
PUT s1=;

s2 = PDF("LOGNORMAL", -1.5, , 2);
PUT s2=;

s3 = PDF("LOGNORMAL", 1.5, 1, 0);
PUT s3=;

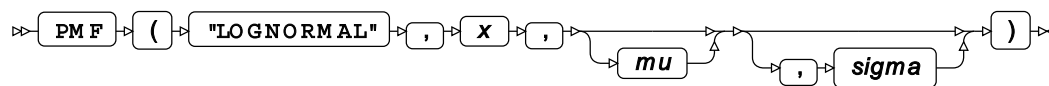
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

PMF – LOGNORMAL

Returns the value of the probability mass function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of PDF – LOGNORMAL.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu)^2}{2\sigma^2}\right)$$

where μ is the mean of the natural logarithm of the random variable and σ is the standard deviation of the natural logarithm of the random variable.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PMF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = PMF("LOGNORMAL", 1.5, 0, 1);
  PUT s1=;
  s2 = PMF("LOGNORMAL", 1.5, 0);
  PUT s2=;
  s3 = PMF("LOGNORMAL", 1.5);
  PUT s3=;

  s4 = PMF("LOGNORMAL", 3, 2.5, 1.2);
  PUT s4=;
  s5 = PMF("LOGNORMAL", 12, 2.5, 1.2);
  PUT s5=;
  s6 = PMF("LOGNORMAL", 40, 2.5, 1.2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.2449736517
s2=0.2449736517
s3=0.2449736517
s4=0.056035057
s5=0.0277021337
s6=0.0050877715
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, PMF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = PMF("LOGNORMAL", 1.5, 1, 2, 3);
  PUT s1=;

  s2 = PMF("LOGNORMAL", -1.5, , 2);
  PUT s2=;

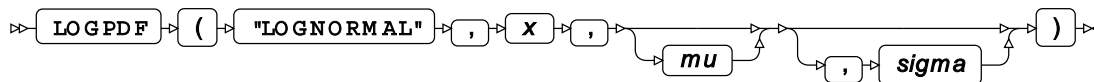
  s3 = PMF("LOGNORMAL", 1.5, 1, 0);
  PUT s3=;

RUN;
```

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

LOGPDF – LOGNORMAL

Returns the value of the natural logarithm of the probability density function at the specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of LOGPMF – LOGNORMAL.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$f(x; \mu, \sigma) = \begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \log \left[\frac{1}{x\sigma\sqrt{2\pi}} \exp \left(\frac{-(\log(x) - \mu)^2}{2\sigma^2} \right) \right]$$

where μ is the mean of the natural logarithm of the random variable and σ is the standard deviation of the natural logarithm of the random variable.

Return type: Numeric

X

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("LOGNORMAL", 1.5, 0, 1);
PUT s1=;
s2 = LOGPDF("LOGNORMAL", 1.5, 0);
PUT s2=;
s3 = LOGPDF("LOGNORMAL", 1.5);
PUT s3=;

s4 = LOGPDF("LOGNORMAL", 3, 2.5, 1.2);
PUT s4=;
s5 = LOGPDF("LOGNORMAL", 12, 2.5, 1.2);
PUT s5=;
s6 = LOGPDF("LOGNORMAL", 40, 2.5, 1.2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=-1.406604618
s2=-1.406604618
s3=-1.406604618
s4=-2.881777767
s5=-3.58624584
s6=-5.280915362
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, LOGPDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("LOGNORMAL", 1.5, 1, 2, 3);
PUT s1=;

s2 = LOGPDF("LOGNORMAL", -1.5, , 2);
PUT s2=;

s3 = LOGPDF("LOGNORMAL", 1.5, 1, 0);
PUT s3=;

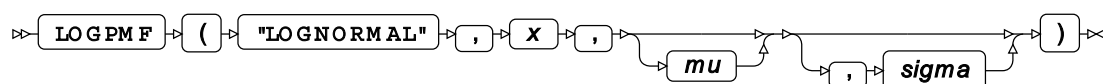
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

LOGPMF – LOGNORMAL

Returns the value of the natural logarithm of the probability mass function at a specified point for the Lognormal distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – LOGNORMAL.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \log \left[\frac{1}{x\sigma\sqrt{2\pi}} \exp \left(-\frac{(\log(x) - \mu)^2}{2\sigma^2} \right) \right]$$

where μ is the mean of the natural logarithm of the random variable and σ is the standard deviation of the natural logarithm of the random variable.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPMF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGPMF("LOGNORMAL", 1.5, 0, 1);
  PUT s1=;
  s2 = LOGPMF("LOGNORMAL", 1.5, 0);
  PUT s2=;
  s3 = LOGPMF("LOGNORMAL", 1.5);
  PUT s3=;

  s4 = LOGPMF("LOGNORMAL", 3, 2.5, 1.2);
  PUT s4=;
  s5 = LOGPMF("LOGNORMAL", 12, 2.5, 1.2);
  PUT s5=;
  s6 = LOGPMF("LOGNORMAL", 40, 2.5, 1.2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=-1.406604618
s2=-1.406604618
s3=-1.406604618
s4=-2.881777767
s5=-3.58624584
s6=-5.280915362
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, LOGPMF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGPMF("LOGNORMAL", 1.5, 1, 2, 3);
  PUT s1=;

  s2 = LOGPMF("LOGNORMAL", -1.5, , 2);
  PUT s2=;

  s3 = LOGPMF("LOGNORMAL", 1.5, 1, 0);
  PUT s3=;

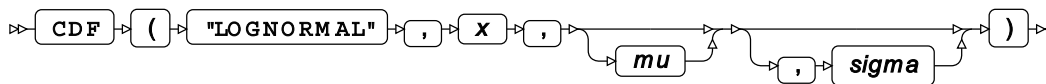
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

CDF – LOGNORMAL

Returns the value of the cumulative density function at the specified point for the Lognormal distribution with the specified mean and standard deviation.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \frac{1}{2} + \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2} \sigma} \right]$$

where:

- $\operatorname{erf}(z)$ is the error function $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt$
- μ is the mean of the natural logarithm of the random variable
- σ is the standard deviation of the natural logarithm of the random variable

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: $\sigma > 0$

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, CDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("LOGNORMAL", 1.5, 0, 1);
PUT s1=;
s2 = CDF("LOGNORMAL", 1.5, 0);
PUT s2=;
s3 = CDF("LOGNORMAL", 1.5);
PUT s3=;

s4 = CDF("LOGNORMAL", 3, 2.5, 1.2);
PUT s4=;
s5 = CDF("LOGNORMAL", 12, 2.5, 1.2);
PUT s5=;
s6 = CDF("LOGNORMAL", 40, 2.5, 1.2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.6574321695
s2=0.6574321695
s3=0.6574321695
s4=0.1214390656
s5=0.4949823193
s6=0.8390919839
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, CDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("LOGNORMAL", 1.5, 1, 2, 3);
PUT s1=;

s2 = CDF("LOGNORMAL", -1.5, , 2);
PUT s2=;

s3 = CDF("LOGNORMAL", 1.5, 1, 0);
PUT s3=;

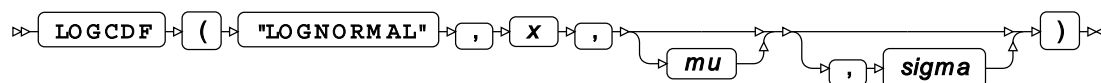
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

LOGCDF – LOGNORMAL

Returns the value of the natural logarithm of the cumulative density function at the specified point for the Lognormal distribution with the specified mean and standard deviation.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \log \left[\frac{1}{2} + \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2} \sigma} \right] \right]$$

where:

- $\operatorname{erf}(z)$ is the error function $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt$
- μ is the mean of the natural logarithm of the random variable
- σ is the standard deviation of the natural logarithm of the random variable

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGCDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGCDF("LOGNORMAL", 1.5, 0, 1);
  PUT s1=;
  s2 = LOGCDF("LOGNORMAL", 1.5, 0);
  PUT s2=;
  s3 = LOGCDF("LOGNORMAL", 1.5);
  PUT s3=;

  s4 = LOGCDF("LOGNORMAL", 3, 2.5, 1.2);
  PUT s4=;
  s5 = LOGCDF("LOGNORMAL", 12, 2.5, 1.2);
  PUT s5=;
  s6 = LOGCDF("LOGNORMAL", 40, 2.5, 1.2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=-0.419413685
s2=-0.419413685
s3=-0.419413685
s4=-2.10834266
s5=-0.703233236
s6=-0.175434943
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, LOGCDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGCDF("LOGNORMAL", 1.5, 1, 2, 3);
  PUT s1=;

  s2 = LOGCDF("LOGNORMAL", -1.5, , 2);
  PUT s2=;

  s3 = LOGCDF("LOGNORMAL", 1.5, 1, 0);
  PUT s3=;

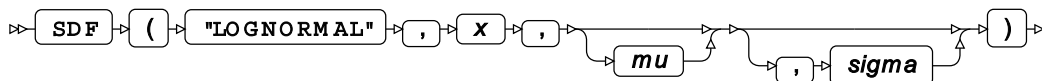
RUN;
```


All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

SDF – LOGNORMAL

Returns the value of the survival function at the specified point for the Lognormal distribution with the specified mean and standard deviation.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \frac{1}{2} - \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2} \sigma} \right]$$

where:

- $\operatorname{erf}(z)$ is the error function $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt$
- μ is the mean of the natural logarithm of the random variable
- σ is the standard deviation of the natural logarithm of the random variable

Return type: Numeric

x

Type: Numeric

The point at which to calculate the value of the survival function.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, SDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("LOGNORMAL", 1.5, 0, 1);
PUT s1=;
s2 = SDF("LOGNORMAL", 1.5, 0);
PUT s2=;
s3 = SDF("LOGNORMAL", 1.5);
PUT s3=;

s4 = SDF("LOGNORMAL", 3, 2.5, 1.2);
PUT s4=;
s5 = SDF("LOGNORMAL", 12, 2.5, 1.2);
PUT s5=;
s6 = SDF("LOGNORMAL", 40, 2.5, 1.2);
PUT s6=;

RUN;
```

This produces the following output:

```
s1=0.3425678305
s2=0.3425678305
s3=0.3425678305
s4=0.8785609344
s5=0.5050176807
s6=0.1609080161
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, SDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("LOGNORMAL", 1.5, 1, 2, 3);
PUT s1=;

s2 = SDF("LOGNORMAL", -1.5, , 2);
PUT s2=;

s3 = SDF("LOGNORMAL", 1.5, 1, 0);
PUT s3=;

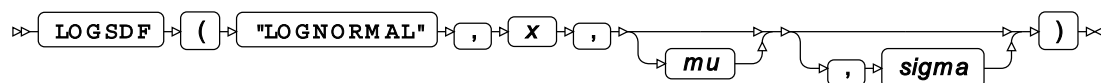
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

LOGSDF – LOGNORMAL

Returns the value of the natural logarithm of the survival function at the specified point for the Lognormal distribution with the specified mean and standard deviation.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for $\sigma > 0$.

The calculated value for the Lognormal distribution is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \mu, \sigma) \end{cases}$$

$$f(x; \mu, \sigma) = \log \left[\frac{1}{2} - \frac{1}{2} \operatorname{erf} \left[\frac{\log(x) - \mu}{\sqrt{2} \sigma} \right] \right]$$

where:

- $\operatorname{erf}(z)$ is the error function $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt$
- μ is the mean of the natural logarithm of the random variable
- σ is the standard deviation of the natural logarithm of the random variable

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGSDF – LOGNORMAL is called for various Lognormal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGSDF("LOGNORMAL", 1.5, 0, 1);
  PUT s1=;
  s2 = LOGSDF("LOGNORMAL", 1.5, 0);
  PUT s2=;
  s3 = LOGSDF("LOGNORMAL", 1.5);
  PUT s3=;

  s4 = LOGSDF("LOGNORMAL", 3, 2.5, 1.2);
  PUT s4=;
  s5 = LOGSDF("LOGNORMAL", 12, 2.5, 1.2);
  PUT s5=;
  s6 = LOGSDF("LOGNORMAL", 40, 2.5, 1.2);
  PUT s6=;

RUN;
```

This produces the following output:

```
s1=-1.071285596
s2=-1.071285596
s3=-1.071285596
s4=-0.129470012
s5=-0.683161839
s6=-1.826922406
```

The first three examples specify the same point ($x = 1.5$) in a Lognormal distribution where the mean of the logarithm of the random variable is 0 and the standard deviation of the logarithm of the random variable is 1. So *s1*, *s2* and *s3* contain the same value.

The remaining three examples specify different points ($x = 3$, $x = 12$ and $x = 40$) in the same Lognormal distribution, where the mean of the logarithm of the random variable is 2.5 and the standard deviation of the logarithm of the random variable is 1.2.

Argument errors

In this example, LOGSDF – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGSDF("LOGNORMAL", 1.5, 1, 2, 3);
  PUT s1=;

  s2 = LOGSDF("LOGNORMAL", -1.5, , 2);
  PUT s2=;

  s3 = LOGSDF("LOGNORMAL", 1.5, 1, 0);
  PUT s3=;

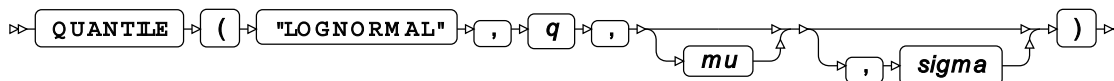
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

QUANTILE – LOGNORMAL

Returns the value of the quantile function at the specified point for the Lognormal distribution with the specified mean and standard deviation.



The Lognormal distribution is a continuous probability distribution where the natural logarithm of the random variable is normally distributed.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*). The mean and standard deviation specified are the mean and standard deviation for the (normally-distributed) natural logarithm of the random variable, not the mean and standard deviation of the random variable itself.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified.

This function is defined for

$$\begin{cases} 0 < q < 1 \\ \sigma > 0 \end{cases}$$

The calculated value, x , for the Lognormal distribution is

$$f(q; \mu, \sigma) = \inf \{x : q \leq \text{CDF}(x; \mu, \sigma)\}$$

where

- $x \in \mathbb{R}$
- $\inf\{x\}$ (*infimum*) is the greatest lower bound of x
- μ is the mean of the (normally distributed) natural logarithm of the random variable
- σ is the standard deviation of the (normally-distributed) natural logarithm of the random variable
- $\text{CDF}(x; \mu, \sigma)$ is the cumulative density function of the Lognormal distribution

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 0

The mean of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

Default: 1

Restriction: *sigma* > 0

The standard deviation of the natural logarithm of the random variable.

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Basic example

In this example, QUANTILE – LOGNORMAL is called for different Lognormal distributions. The returned values are then passed to CDF – LOGNORMAL for the same Lognormal distributions. The results are written to the log.

The results are written to the log.

```
DATA _NULL_;

s1 = QUANTILE("LOGNORMAL", 0.25, 2.5, 1.2);
s2 = QUANTILE("LOGNORMAL", 0.75, 2.5, 1.2);
PUT s1= s2=;

s3 = CDF("LOGNORMAL", s1, 2.5, 1.2);
s4 = CDF("LOGNORMAL", s2, 2.5, 1.2);
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=5.422800068 s2=27.368362698
s3=0.25 s4=0.75
```

Variable *s1*=5.422800068 is the value returned by QUANTILE – LOGNORMAL for *q*=0.25 for a Lognormal distribution with mean 2.5 and standard deviation 1.2.

Variable *s2*=27.368362698 is the value returned by QUANTILE – LOGNORMAL for *q*=0.75.

As a comparison, variables *s3* and *s4* contain the results of passing *s1* and *s2* to CDF – LOGNORMAL. As QUANTILE – LOGNORMAL and CDF – LOGNORMAL are inverse functions, the values returned from CDF – LOGNORMAL are the same as the values originally passed to QUANTILE – LOGNORMAL.

Argument errors

In this example, QUANTILE – LOGNORMAL is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = QUANTILE("LOGNORMAL", 0.25, 2.5, 1.2, 3);
PUT s1=;

s2 = QUANTILE("LOGNORMAL", 0.25, , 1.2);
PUT s2=;

s3 = QUANTILE("LOGNORMAL", 0.25, 2.5, 0);
PUT s3=;

s4 = QUANTILE("LOGNORMAL", 0, 2.5, 1.2);
PUT s4=;

s5 = QUANTILE("LOGNORMAL", -1, 2.5, 1.2);
PUT s5=;

RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example specifies *sigma* but not *mu*. The third example specifies an invalid value for *sigma*.

The fourth and fifth examples specify invalid values for *q* (*q*=0 and *q*=-1 respectively).

RAND – LOGNORMAL

Returns a random number from the Lognormal distribution.

```
➤ RAND ➤ ( ➤ "LOGNORMAL" ➤ ) ➤
```

The distribution is parameterised using a mean of 0 and a standard deviation of 1.

This function does not take any variable arguments.

Each time you execute this function within a DATA step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

The return value is positive.

Example

In this example, a random number from the Lognormal distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("LOGNORMAL");
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.5024141225
0.3646048429
1.1400518351
4.1975318463
0.1907837004
```

Running the `DATA` step again produces the following output.

```
The random numbers are:
0.1353004462
0.6602126687
2.9420670479
1.2557263294
0.7885070773
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  PUT "The random numbers are:";
  CALL STREAMINIT(9);
  DO i = 1 TO 5;
    result = RAND("LOGNORMAL");
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.7506445418
0.5203794406
2.9996140074
1.6572896735
0.1503519974
```

Running the DATA step again produces the same output.

Negative Binomial distribution

Functions for the Negative Binomial distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – NEGBINOMIAL [↗](#)..... 1158

$$\binom{k+r-1}{k} p^k (1-p)^r$$

Returns the probability density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of PMF – NEGBINOMIAL.

PMF – NEGBINOMIAL [↗](#)..... 1159

$$\binom{k+r-1}{k} p^k (1-p)^r$$

Returns the probability mass of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of PDF – NEGBINOMIAL.

LOGPDF – NEGBINOMIAL [↗](#)..... 1161

$$\log \left[\binom{k+r-1}{k} p^k (1-p)^r \right]$$

Returns the natural logarithm of the probability density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of LOGPMF – NEGBINOMIAL.

LOGPMF – NEGBINOMIAL [↗](#)..... 1162

$$\log \left[\binom{k+r-1}{k} p^k (1-p)^r \right]$$

Returns the natural logarithm of the probability mass of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of LOGPDF – NEGBINOMIAL.

CDF – NEGBINOMIAL [↗](#)..... 1164

$$I_p(k, r+1) = \frac{B(p; k, r+1)}{B(k, r+1)}$$

Returns the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is similar to PROBNEGB.

PROBNEGB [↗](#)..... 1166

$$I_p(k, r+1) = \frac{B(p; k, r+1)}{B(k, r+1)}$$

Returns the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is similar to CDF – NEGBINOMIAL.

LOGCDF – NEGBINOMIAL [↗](#)..... 1167

$$\log I_p(k, r+1)$$

Returns the natural logarithm of the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

SDF – NEGBINOMIAL [↗](#)..... 1169

$$1 - I_p(k, r+1)$$

Returns the survival of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

LOGSDF – NEGBINOMIAL [↗](#)..... 1171

$$\log [1 - I_p(k, r+1)]$$

Returns the natural logarithm of the survival of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

QUANTILE – NEGBINOMIAL [↗](#)..... 1172

$$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$$

Returns the quantile of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

RAND – NEGBINOMIAL [↗](#)..... 1174

Returns a random number from the Negative Binomial distribution based on the probability of success and the number of successes in Bernoulli trials.

PDF – NEGBINOMIAL

Returns the probability density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of PMF – NEGBINOMIAL.

⇒ PDF ("NEGBINOMIAL" , *r* , *p* , *k*) ⇒

Calculates the probability density function for the Negative Binomial distribution for the number of failures *r*, based on the probability of success *p* and the number of successes *k* in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned}
 &0 \leq p \leq 1 \\
 &r \in \mathbb{Z} \\
 &k > 0, k \in \mathbb{Z} \\
 &\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p; k, r) \end{cases} \\
 &f(p; k, r) = \binom{k+r-1}{k} p^k (1-p)^r \quad \text{where} \quad \binom{k+r-1}{k} = \frac{(k+r-1)!}{k!(r-1)!}
 \end{aligned}$$

Return type: Numeric

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: *r* must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: *k* > 0 and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1;
  s2 = PDF("NEGBINOMIAL", -1, 0.7, 8);
  PUT s2;
  s3 = PDF("NEGBINOMIAL", 10, 0.7, 8);
  PUT s3;
  s4 = PDF("NEGBINOMIAL", 0.6, 0.7, 8);
  PUT s4;
  s5 = PDF("NEGBINOMIAL", 0, -1.7, 8);
  PUT s5;
RUN;
```

This produces the following output:

```
s1=0.05764801
s2=0
s3=0.0066202107
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

PMF – NEGBINOMIAL

Returns the probability mass of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of PDF – NEGBINOMIAL.

PMF (**"NEGBINOMIAL"** , ***r*** , ***p*** , ***k***)

Calculates the probability mass function for the Negative Binomial distribution for the number of failures r , based on the probability of success p and the number of successes k in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ r &\in \mathbb{Z} \\ k &> 0, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p; k, r) \end{cases}$$

$$f(p; k, r) = \binom{k+r-1}{k} p^k (1-p)^r \text{ where } \binom{k+r-1}{k} = \frac{(k+r-1)!}{k!(r-1)!}$$

Return type: Numeric

r**Type:** Numeric

The number of failures until the specified number of successes has been reached.

Restriction: r must be integer

If the argument is out of range, a missing value is returned.

p**Type:** Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k**Type:** Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = PMF("NEGBINOMIAL", 0, 0.7, 8);  
PUT s1=;  
s2 = PMF("NEGBINOMIAL", -1, 0.7, 8);  
PUT s2=;  
s3 = PMF("NEGBINOMIAL", 10, 0.7, 8);  
PUT s3=;  
s4 = PMF("NEGBINOMIAL", 0.6, 0.7, 8);  
PUT s4=;  
s5 = PMF("NEGBINOMIAL", 0, -1.7, 8);  
PUT s5=;  
RUN;
```

This produces the following output:

```
s1=0.05764801  
s2=0  
s3=0.0066202107  
s4=.  
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGPDF – NEGBINOMIAL

Returns the natural logarithm of the probability density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of LOGPMF – NEGBINOMIAL.

LOGPDF ("NEGBINOMIAL" , *r* , *p* , *k*)

Calculates the natural logarithm of the probability density function for the Negative Binomial distribution for the number of failures *r*, based on the probability of success *p* and the number of successes *k* in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} &0 \leq p \leq 1 \\ &r \geq 0, r \in \mathbb{Z} \\ &\text{if } p = 0 \text{ or } p = 1, \text{ then } r = 0 \\ &k > 0, k \in \mathbb{Z} \\ &\begin{cases} \text{if } p = 0 \text{ and } r = 0 & \text{return } 0 \\ \text{if } p = 1 \text{ and } r = 0 & \text{return } 0 \\ \text{if } 0 < p < 1 & \text{return } f(p; k, r) \end{cases} \end{aligned}$$

$$f(p; k, r) = \log \left[\binom{k+r-1}{k} p^k (1-p)^r \right] = \log \binom{k+r-1}{k} + k \log p + r \log (1-p)$$

$$\text{where } \binom{k+r-1}{k} = \frac{(k+r-1)!}{k!(r-1)!}$$

Return type: Numeric

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restrictions:

- $r \geq 0$ must be integer
- if $p = 0$ or $p = 1$, then $r = 0$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k **Type:** Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = LOGPDF ("NEGBINOMIAL", -1, 0.7, 8);
  PUT s2=;
  s3 = LOGPDF ("NEGBINOMIAL", 0, -1.7, 8);
  PUT s3=;
RUN;
```

This produces the following output:

```
s1=-2.853399552
s2=.
s3=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGPMF – NEGBINOMIAL

Returns the natural logarithm of the probability mass of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is an alias of LOGPDF – NEGBINOMIAL.

```
>> LOGPMF ( "NEGBINOMIAL" , , r , , p , , k ) <<
```

Calculates the natural logarithm of the probability mass function for the Negative Binomial distribution for the number of failures r , based on the probability of success p and the number of successes k in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned}
 &0 \leq p \leq 1 \\
 &r \geq 0, r \in \mathbb{Z} \\
 &\text{if } p = 0 \text{ or } p = 1, \text{ then } r = 0 \\
 &k > 0, k \in \mathbb{Z} \\
 &\begin{cases} \text{if } p = 0 \text{ and } r = 0 & \text{return } 0 \\ \text{if } p = 1 \text{ and } r = 0 & \text{return } 0 \\ \text{if } 0 < p < 1 & \text{return } f(p; k, r) \end{cases} \\
 &f(p; k, r) = \log \left[\binom{k+r-1}{k} p^k (1-p)^r \right] = \log \binom{k+r-1}{k} + k \log p + r \log (1-p) \\
 &\text{where } \binom{k+r-1}{k} = \frac{(k+r-1)!}{k!(r-1)!}
 \end{aligned}$$

Return type: Numeric

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restrictions:

- $r \geq 0$ must be integer
- if $p = 0$ or $p = 1$, then $r = 0$

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF ("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = LOGPMF ("NEGBINOMIAL", -1, 0.7, 8);
  PUT s2=;
  s3 = LOGPMF ("NEGBINOMIAL", 0, -1.7, 8);
  PUT s3=;
RUN;
```

This produces the following output:

```
s1=-2.853399552
s2=.
s3=.
```

The last two examples return a missing value because one of the arguments is out of range.

CDF – NEGBINOMIAL

Returns the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is similar to PROBNEGB.

➤ CDF ➤ (➤ "NEGBINOMIAL" ➤ , ➤ *r* ➤ , ➤ *p* ➤ , ➤ *k* ➤) ➤

Calculates the cumulative density function for the Negative Binomial distribution for the number of failures *r*, based on the probability of success *p* and the number of successes *k* in Bernoulli trials.

This function is defined under the following conditions:

$$0 \leq p \leq 1$$

$$r \in \mathbb{Z}$$

$$k > 0, k \in \mathbb{Z}$$

$$\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } I_p(k, r+1) \end{cases}$$

$$I_p(k, r+1) = \frac{B(p; k, r+1)}{B(k, r+1)}$$

$$B(p; k, r+1) = \int_0^p t^{k-1} (1-t)^r dt$$

$$B(k, r+1) = B(p; k, r+1) \big|_{p=1}$$

where $I_p(k, r+1)$ is the regularised incomplete Beta function; $B(p; k, r+1)$ is the incomplete Beta function; and $B(k, r+1)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: *r* must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: *k* > 0 and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = CDF("NEGBINOMIAL",0,0.7,8);  
  PUT s1=;  
  s2 = CDF("NEGBINOMIAL",-1,0.7,8);  
  PUT s2=;  
  s3 = CDF("NEGBINOMIAL",10,0.7,8);  
  PUT s3=;  
  s4 = CDF("NEGBINOMIAL",0.6,0.7,8);  
  PUT s4=;  
  s5 = CDF("NEGBINOMIAL",0,-1.7,8);  
  PUT s5=;  
RUN;
```

This produces the following output:

```
s1=0.05764801  
s2=0  
s3=0.993927486  
s4=.  
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

PROBNEGB

Returns the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials. This function is similar to CDF – NEGBINOMIAL.

⇒ PROBNEGB ((p , k , r)) ⇒

Note:

Function PROBNEGB differs from CDF ("NEGBINOMIAL", r, p, k) in the order of the arguments and in the restrictions imposed on k and r .

Calculates the cumulative density function for the Negative Binomial distribution for the number of failures r , based on the probability of success p and the number of successes k in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ r &\geq 0, r \in \mathbb{Z} \\ k &> 0, k \in \mathbb{Z} \end{aligned}$$

$$I_p(k, r+1) = \frac{B(p; k, r+1)}{B(k, r+1)}$$

$$B(p; k, r+1) = \int_0^p t^{k-1} (1-t)^r dt$$

$$B(k, r+1) = B(p; k, r+1) \big|_{p=1}$$

where $I_p(k, r+1)$ is the regularised incomplete Beta function; $B(p; k, r+1)$ is the incomplete Beta function; and $B(k, r+1)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: $r \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PROBNEGB(0.7,8,0);
  PUT s1=;
  s2 = PROBNEGB(0.7,8,-1);
  PUT s2=;
  s3 = PROBNEGB(-1.7,8,0);
  PUT s3=;
RUN;
```

This produces the following output:

```
s3=0.05764801
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGCDF – NEGBINOMIAL

Returns the natural logarithm of the cumulative density of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

```
LOGCDF ( "NEGBINOMIAL" , r , p , k )
```

Calculates the natural logarithm of the cumulative density function for the Negative Binomial distribution for the number of failures r , based on the probability of success p and the number of successes k in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 < p &\leq 1 \\ r &\geq 0, r \in \mathbb{Z} \\ k &> 0, k \in \mathbb{Z} \end{aligned}$$

$$f(p; k, r) = \log I_p(k, r+1) = \log B(p; k, r+1) - \log B(k, r+1)$$

$$B(p; k, r+1) = \int_0^p t^{k-1} (1-t)^r dt$$

$$B(k, r+1) = B(p; k, r+1) \Big|_{p=1}$$

where $I_p(k, r+1)$ is the regularised incomplete Beta function; $B(p; k, r+1)$ is the incomplete Beta function; and $B(k, r+1)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is negative or zero.

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: $r \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 < p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF ("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = LOGCDF ("NEGBINOMIAL", 10, 0.7, 8);
  PUT s2=;
  s3 = LOGCDF ("NEGBINOMIAL", 0.6, 0.7, 8);
  PUT s3=;
  s4 = LOGCDF ("NEGBINOMIAL", 0, -1.7, 8);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=-2.853399552
s2=-0.006091027
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

SDF – NEGBINOMIAL

Returns the survival of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

➤ SDF ➤ (➤ "NEGBINOMIAL" ➤ , ➤ *r* ➤ , ➤ *p* ➤ , ➤ *k* ➤) ➤ ➤

Calculates the survival, or the complement to the cumulative density function, for the Negative Binomial distribution for the number of failures *r*, based on the probability of success *p* and the number of successes *k* in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p \leq 1 \\ r &\in \mathbb{Z} \\ k &> 0, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } r < 0 & \text{return } 1 \\ \text{otherwise} & \text{return } f(p; k, r+1) \end{cases}$$

$$f(p; k, r) = 1 - I_p(k, r+1) = 1 - \frac{B(p; k, r+1)}{B(k, r+1)}$$

$$B(p; k, r+1) = \int_0^p t^{k-1} (1-t)^r dt$$

$$B(k, r+1) = B(p; k, r+1) \big|_{p=1}$$

where $I_p(k, r+1)$ is the regularised incomplete Beta function; $B(p; k, r+1)$ is the incomplete Beta function; and $B(k, r+1)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is less than or equal to one.

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: *r* must be integer

If the argument is out of range, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: *k* > 0 and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = SDF("NEGBINOMIAL", -1, 0.7, 8);
  PUT s2=;
  s3 = SDF("NEGBINOMIAL", 10, 0.7, 8);
  PUT s3=;
  s4 = SDF("NEGBINOMIAL", 0.6, 0.7, 8);
  PUT s4=;
  s5 = SDF("NEGBINOMIAL", 0, -1.7, 8);
  PUT s5=;
RUN;
```


This produces the following output:

```
s1=0.94235199
s2=1
s3=0.006072514
s4=.
s5=.
```

The last two examples return a missing value because one of the arguments is out of range.

LOGSDF – NEGBINOMIAL

Returns the natural logarithm of the survival of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

➤ LOGSDF ➤ (➤ "NEGBINOMIAL" ➤ , ➤ *r* ➤ , ➤ *p* ➤ , ➤ *k* ➤) ➤ ➤

Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Negative Binomial distribution for the number of failures *r*, based on the probability of success *p* and the number of successes *k* in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq p < 1 \\ r &\geq 0, r \in \mathbb{Z} \\ k &> 0, k \in \mathbb{Z} \end{aligned}$$

$$\begin{cases} \text{if } r < 0 & \text{return } 0 \\ \text{otherwise} & \text{return } f(p; k, r+1) \end{cases}$$

$$f(p; k, r) = \log[1 - I_p(k, r+1)] = \log\left[1 - \frac{B(p; k, r+1)}{B(k, r+1)}\right]$$

$$B(p; k, r+1) = \int_0^p t^{k-1} (1-t)^r dt$$

$$B(k, r+1) = B(p; k, r+1) \Big|_{p=1}$$

where $I_p(k, r+1)$ is the regularised incomplete Beta function; $B(p; k, r+1)$ is the incomplete Beta function; and $B(k, r+1)$ is the Beta function, see [BETA](#) (page 1819).

Return type: Numeric

The return value is negative or zero.

r

Type: Numeric

The number of failures until the specified number of successes has been reached.

Restriction: *r* must be integer

If the argument is out of range, a missing value is returned.

p **Type:** Numeric

The probability of success in all the trials.

Restriction: $0 \leq p < 1$

If the argument is out of range, a missing value is returned.

 k **Type:** Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = LOGSDF ("NEGBINOMIAL", -1, 0.7, 8);  
PUT s1=;  
s2 = LOGSDF ("NEGBINOMIAL", 0, 0.7, 8);  
PUT s2=;  
s3 = LOGSDF ("NEGBINOMIAL", 0.6, 0.7, 8);  
PUT s3=;  
s4 = LOGSDF ("NEGBINOMIAL", 0, -1.7, 8);  
PUT s4=;  
RUN;
```

This produces the following output:

```
s1=0  
s2=-0.059376412  
s3=.  
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

QUANTILE – NEGBINOMIAL

Returns the quantile of the Negative Binomial distribution for a specified number of failures based on the probability of success and the number of successes in Bernoulli trials.

➤ **QUANTILE** ➤ (➤ **"NEGBINOMIAL"** ➤ , ➤ **q** ➤ , ➤ **p** ➤ , ➤ **k** ➤) ➤

Calculates the quantile x , or the inverse of the cumulative density function, for the Negative Binomial distribution for probability value q based on the probability of success p and the number of successes k in Bernoulli trials.

This function is defined under the following conditions:

$$\begin{aligned} 0 &\leq q < 1 \\ 0 &< p \leq 1 \\ k &> 0, k \in \mathbb{Z} \\ \begin{cases} \text{if } q=0 & \text{return } 0 \\ \text{if } q=1 & \text{return } 1 \\ \text{otherwise} & \text{return } f(q) \end{cases} \\ f(q) &= \inf \{x: q \leq \text{CDF}(x)\} \end{aligned}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function for the Negative Binomial distribution, see section *CDF – NEGBINOMIAL* [↗](#) (page 1164).

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 \leq q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 < p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: $k > 0$ and must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Negative Binomial distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = QUANTILE("NEGBINOMIAL", 0, 0.7, 8);
  PUT s1=;
  s2 = QUANTILE("NEGBINOMIAL", 0.6, 0.7, 8);
  PUT s2=;
  s3 = QUANTILE("NEGBINOMIAL", 0.6, 0, 8);
  PUT s3=;
  s4 = QUANTILE("NEGBINOMIAL", 0, -1.7, 8);
  PUT s4=;
RUN;
```

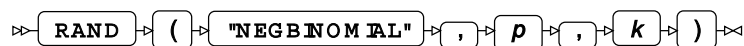
This produces the following output:

```
s1=0
s2=4
s3=.
s4=.
```

The last two examples return a missing value because one of the arguments is out of range.

RAND – NEGBINOMIAL

Returns a random number from the Negative Binomial distribution based on the probability of success and the number of successes in Bernoulli trials.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS)* - Special issue on uniform random number generation 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

p

Type: Numeric

The probability of success in all the trials.

Restriction: $0 \leq p \leq 1$

If the argument is out of range, a missing value is returned.

k

Type: Numeric

The number of successes.

Restriction: $k \geq 1$

If the argument is out of range, a missing value is returned.

Example

In this example, a random number from the Negative Binomial distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("NEGBINOMIAL", 0.79,100);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
66.386846036  
73.391732883  
68.269978958  
72.467947197  
54.952402765
```

Running the DATA step again produces the following output.

```
The random numbers are:  
62.835033565  
71.090153203  
66.894372987  
55.933535406  
65.669639433
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  CALL STREAMINIT(9);  
  DO i = 1 TO 5;  
    result = RAND("NEGBINOMIAL", 0.79,100);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:
63.26274407
69.849910723
61.807560636
64.036333375
73.657909167
```

Running the `DATA` step again produces the same output.

Normal distribution

Functions and `CALL` routines for the Normal distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – NORMAL [↗](#)..... 1178

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Returns the value of the probability density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of `PMF – NORMAL`, `PDF – GAUSSIAN` and `PMF – GAUSSIAN`.

PMF – NORMAL [↗](#)..... 1181

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

Returns the value of the probability mass function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of `PDF – NORMAL`, `PDF – GAUSSIAN` and `PMF – GAUSSIAN`.

LOGPDF – NORMAL [↗](#)..... 1183

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

Returns the value of the natural logarithm of the probability density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of `LOGPMF – NORMAL`, `LOGPDF – GAUSSIAN` and `LOGPMF – GAUSSIAN`.

LOGPMF – NORMAL [↗](#)..... 1185

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

Returns the value of the natural logarithm of the probability mass function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – NORMAL, LOGPDF – GAUSSIAN and LOGPMF – GAUSSIAN.

CDF – NORMAL [↗](#)..... 1187

$$\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^x\exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)dt$$

Returns the value of the cumulative density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of CDF – GAUSSIAN and is similar to PROBNORM.

LOGCDF – NORMAL [↗](#).....1189

$$\log\left(\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^x\exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)dt\right)$$

Returns the value of the natural logarithm of the cumulative density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGCDF – GAUSSIAN.

SDF – NORMAL [↗](#)..... 1192

$$1-\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^x\exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)dt$$

Returns the value of the survival function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of SDF – GAUSSIAN.

LOGSDF – NORMAL [↗](#).....1194

$$\log\left(1-\frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^x\exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)dt\right)$$

Returns the value of the natural logarithm of the survival function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGSDF – GAUSSIAN.

QUANTILE – NORMAL [↗](#).....1196

$$\inf\{x: q \leq \text{CDF}(x; \mu, \sigma)\}$$

Returns the value of the quantile function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of QUANTILE – GAUSSIAN and is similar to PROBIT.

PROBNORM [↗](#)..... 1198

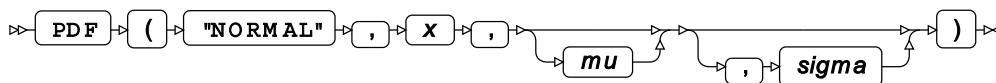
$$\frac{1}{\sqrt{2\pi}}\int_{-\infty}^x\exp\left(-\frac{1}{2}t^2\right)dt$$

Returns the cumulative density function at a given point for the standard Normal distribution with mean 0 and standard deviation 1. This function is similar to CDF – NORMAL and CDF – GAUSSIAN, with the mean and standard deviation set to the default values.

PROBIT ↗	1200
$\inf \{x: q \leq \text{CDF}(x)\}$	
Returns the quantile function at a given point for the standard Normal distribution with mean 0 and standard deviation 1. This function is similar to QUANTILE – NORMAL and QUANTILE – GAUSSIAN with mean and standard deviation set to the default values.	
DEVIANCE – NORMAL ↗	1202
$(x - \mu)^2$	
Returns the deviance of the Normal distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – GAUSSIAN.	
RAND – NORMAL ↗	1202
Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is similar to RANNOR, NORMAL and CALL RANNOR.	
RANNOR ↗	1204
Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is an alias of NORMAL. This function is similar to RAND – NORMAL and CALL RANNOR.	
NORMAL ↗	1206
Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is an alias of RANNOR. This function is similar to RAND – NORMAL and CALL RANNOR.	
CALL RANNOR ↗	1207
Returns a random number from the Normal distribution. This routine is similar to function RAND – NORMAL, NORMAL and RANNOR.	

PDF – NORMAL

Returns the value of the probability density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of PMF – NORMAL, PDF – GAUSSIAN and PMF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the probability density function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: *sigma* > 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the probability density function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = PDF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = PDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = PDF("NORMAL", 0);
  PUT s3=;
  s4 = PDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = PDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = PDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = PDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = PDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=0.3989422804
s2=0.3989422804
s3=0.3989422804
s4=0.0647587978
s5=0.0647587978
s6=.
s7=.
s8=.
```

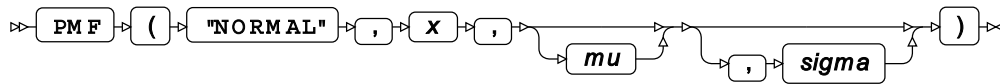
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the probability density function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the probability density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

PMF – NORMAL

Returns the value of the probability mass function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of PDF – NORMAL, PDF – GAUSSIAN and PMF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the probability mass function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: $\sigma > 0$

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the probability mass function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = PMF("NORMAL", 0, 0, 1);
  PUT s1;
  s2 = PMF("NORMAL", 0, 0);
  PUT s2;
  s3 = PMF("NORMAL", 0);
  PUT s3;
  s4 = PMF("NORMAL", -3, 0, 2);
  PUT s4;
  s5 = PMF("NORMAL", 7, 10, 2);
  PUT s5;
  s6 = PMF("NORMAL", -3, 0, 2, 3);
  PUT s6;
  s7 = PMF("NORMAL", -3, , 2);
  PUT s7;
  s8 = PMF("NORMAL", -3, 0, 0);
  PUT s8;
RUN;
```

This produces the following output:

```
s1=0.3989422804
s2=0.3989422804
s3=0.3989422804
s4=0.0647587978
s5=0.0647587978
s6=.
s7=.
s8=.
```

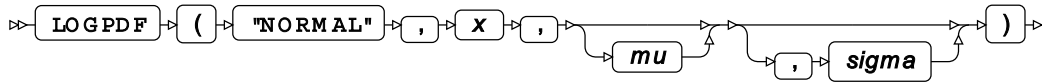
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the probability mass function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the probability mass function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

LOGPDF – NORMAL

Returns the value of the natural logarithm of the probability density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGPMF – NORMAL, LOGPDF – GAUSSIAN and LOGPMF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the probability density function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: $\sigma > 0$

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the probability density function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = LOGPDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = LOGPDF("NORMAL", 0);
  PUT s3=;
  s4 = LOGPDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = LOGPDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = LOGPDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGPDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = LOGPDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.918938533
s2=-0.918938533
s3=-0.918938533
s4=-2.737085714
s5=-2.737085714
s6=.
s7=.
s8=.
```

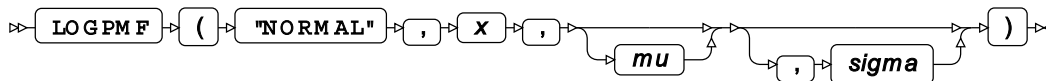
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the probability density function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the probability density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

LOGPMF – NORMAL

Returns the value of the natural logarithm of the probability mass function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGPDF – NORMAL, LOGPDF – GAUSSIAN and LOGPMF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither μ nor σ is specified, μ defaults to 0 and σ defaults to 1, and this function returns the value of the natural logarithm of the probability mass function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: *sigma* > 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the probability mass function of the Normal distribution is calculated for various values of *x*, *mu* and *sigma*. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = LOGPMF("NORMAL", 0, 0);
  PUT s2=;
  s3 = LOGPMF("NORMAL", 0);
  PUT s3=;
  s4 = LOGPMF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = LOGPMF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = LOGPMF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGPMF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = LOGPMF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.918938533
s2=-0.918938533
s3=-0.918938533
s4=-2.737085714
s5=-2.737085714
s6=.
s7=.
s8=.
```

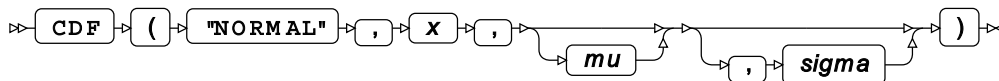
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the probability mass function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the probability mass function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

CDF – NORMAL

Returns the value of the cumulative density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of CDF – GAUSSIAN and is similar to PROBNORM.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the cumulative density function for the standard Normal distribution.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x

Type: Numeric

The point at which to calculate the cumulative density.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: *sigma* > 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the cumulative density function of the Normal distribution is calculated for various values of *x*, *mu* and *sigma*. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = CDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = CDF("NORMAL", 0);
  PUT s3=;
  s4 = CDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = CDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = CDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = CDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = CDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=0.5
s2=0.5
s3=0.5
s4=0.0668072013
s5=0.0668072013
s6=.
s7=.
s8=.
```

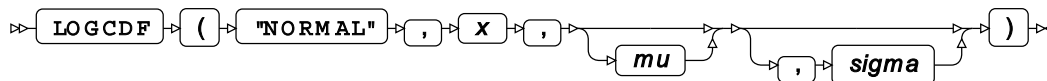
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the cumulative density function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the cumulative density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

LOGCDF – NORMAL

Returns the value of the natural logarithm of the cumulative density function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGCDF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the cumulative density function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log \left(\frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} \left(\frac{t - \mu}{\sigma} \right)^2 \right) dt \right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log\left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: *sigma* > 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the cumulative density function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = LOGCDF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = LOGCDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = LOGCDF("NORMAL", 0);
  PUT s3=;
  s4 = LOGCDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = LOGCDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = LOGCDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGCDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = LOGCDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.693147181
s2=-0.693147181
s3=-0.693147181
s4=-2.705944401
s5=-2.705944401
s6=.
s7=.
s8=.
```

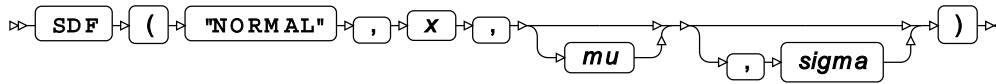
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the cumulative density function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the cumulative density function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, and the eighth example specifies an invalid value for *sigma*.

SDF – NORMAL

Returns the value of the survival function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of SDF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the survival function for the standard Normal distribution at point *x*.

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = 1 - \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right) dt$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x

Type: Numeric

The point at which to calculate the value of the survival function.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: $\sigma > 0$

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the survival function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = SDF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = SDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = SDF("NORMAL", 0);
  PUT s3=;
  s4 = SDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = SDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = SDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = SDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = SDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=0.5
s2=0.5
s3=0.5
s4=0.9331927987
s5=0.9331927987
s6=.
s7=.
s8=.
```

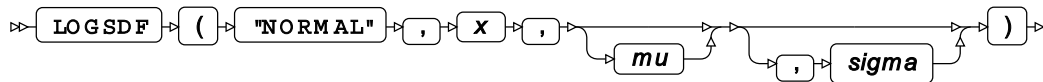
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the survival function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the survival function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

LOGSDF – NORMAL

Returns the value of the natural logarithm of the survival function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of LOGSDF – GAUSSIAN.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the natural logarithm of the survival function for the standard Normal distribution at point x .

This function is defined for $\sigma > 0$.

The calculated value for the Normal distribution with mean μ and standard deviation σ is

$$f(x; \mu, \sigma) = \log \left(1 - \frac{1}{\sigma \sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} \left(\frac{t - \mu}{\sigma} \right)^2 \right) dt \right)$$

and the calculated value for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(x) = \log \left(1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp \left(-\frac{1}{2} t^2 \right) dt \right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: $\sigma > 0$

If σ is specified, μ must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the natural logarithm of the survival function of the Normal distribution is calculated for various values of x , μ and σ . The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSF("NORMAL", 0, 0, 1);
  PUT s1=;
  s2 = LOGSDF("NORMAL", 0, 0);
  PUT s2=;
  s3 = LOGSDF("NORMAL", 0);
  PUT s3=;
  s4 = LOGSDF("NORMAL", -3, 0, 2);
  PUT s4=;
  s5 = LOGSDF("NORMAL", 7, 10, 2);
  PUT s5=;
  s6 = LOGSDF("NORMAL", -3, 0, 2, 3);
  PUT s6=;
  s7 = LOGSDF("NORMAL", -3, , 2);
  PUT s7=;
  s8 = LOGSDF("NORMAL", -3, 0, 0);
  PUT s8=;
RUN;
```

This produces the following output:

```
s1=-0.693147181
s2=-0.693147181
s3=-0.693147181
s4=-0.069143456
s5=-0.069143456
s6=.
s7=.
s8=.
```

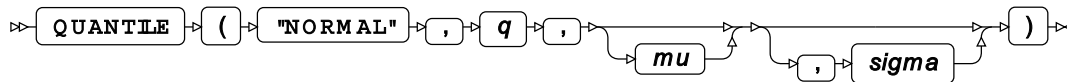
The first three examples all specify the same point, 0, in a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the natural logarithm of the survival function.

The fourth example specifies a point, -3, in a Normal distribution with mean 0 and standard deviation 2. The fifth example specifies a point, 7, in a Normal distribution with mean 10 and standard deviation 2. Although the distribution means are different, both these points are 1.5 standard deviations below the mean, so both examples return the same value for the natural logarithm of the survival function.

The last three examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies σ but not μ , and the eighth example specifies an invalid value for σ .

QUANTILE – NORMAL

Returns the value of the quantile function at a given point for the Normal distribution with the specified mean and standard deviation. This function is an alias of QUANTILE – GAUSSIAN and is similar to PROBIT.



The Normal distribution is also known as the Gaussian distribution.

You can optionally specify the mean μ (*mu*) and standard deviation σ (*sigma*) of the distribution.

If not specified, *mu* defaults to 0 and *sigma* defaults to 1. If *sigma* is specified, *mu* must also be specified. If neither *mu* nor *sigma* is specified, this function returns the value of the quantile function for the standard Normal distribution at *q*.

This function is defined for

$$\begin{cases} 0 < q < 1 \\ \sigma > 0 \end{cases}$$

The calculated value, x , for the Normal distribution with mean μ and standard deviation σ , is:

$$f(q; \mu, \sigma) = \inf \{x: q \leq \text{CDF}(x; \mu, \sigma)\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ (*infinium*) is the greatest lower bound of x , and $\text{CDF}(x, \mu, \sigma)$ is the cumulative density function of the Normal distribution with mean μ and standard deviation σ .

The calculated value, x , for the standard Normal distribution, where $\mu = 0$ and $\sigma = 1$, is

$$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ is the greatest lower bound of x , and $\text{CDF}(x)$ is the cumulative density function of the standard Normal distribution.

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

The mean of the distribution.

Default: 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

sigma

Optional argument

Type: Numeric

The standard deviation of the distribution.

Default: 1

Restriction: *sigma* > 0

If *sigma* is specified, *mu* must also be specified; otherwise a missing value is returned.

If the argument is out of range, a missing value is returned.

Example

In this example, the value of the quantile function of the Normal distribution is calculated for various values of *q*, *mu* and *sigma*. The results are written to the log.

As this function is the inverse of the cumulative density function, this example demonstrates the relationship by using the same points in the distribution as the cumulative density function example, *CDF – NORMAL* [↗](#) (page 1187). Therefore the *q* values in this example are the returned values in the cumulative density function example, and the returned values in this example are the *x* values in the cumulative density function example (subject to rounding errors).

```
DATA _NULL_;
  s1 = QUANTILE("NORMAL", 0.5, 0, 1);
  PUT s1=;
  s2 = QUANTILE("NORMAL", 0.5, 0);
  PUT s2=;
  s3 = QUANTILE("NORMAL", 0.5);
  PUT s3=;
  s4 = QUANTILE("NORMAL", 0.0668072013, 0, 2);
  PUT s4=;
  s5 = QUANTILE("NORMAL", 0.0668072013, 10, 2);
  PUT s5=;
  s6 = QUANTILE("NORMAL", 0.0668072013, 0, 2, 3);
  PUT s6=;
  s7 = QUANTILE("NORMAL", 0.0668072013, , 2);
  PUT s7=;
  s8 = QUANTILE("NORMAL", 0.0668072013, 0, 0);
  PUT s8=;
  s9 = QUANTILE("NORMAL", 1.0, 0, 1);
  PUT s9=;
RUN;
```

This produces the following output:

```
s1=0
s2=0
s3=0
s4=-3
s5=7.0000000005
s6=.
s7=.
s8=.
s9=.
```

The first three examples all specify the same cumulative density value, 0.5, for a standard Normal distribution with mean 0 and standard deviation 1. So they all return the same value for the quantile function. In this case, the value is the point in the distribution where the probability is 0.5 that a random member of the distribution falls below this point. For the standard Normal distribution, this value is the mean, 0.

The fourth example specifies a cumulative density value of 0.0668072013, a mean of 0 and a standard deviation of 2. The fifth example specifies the same cumulative density value and standard deviation, but has a mean of 10. For a Normal distribution with a standard deviation of 2, 0.0668072013 is the value of the cumulative density function at a point 1.5 standard deviations below the mean. So both these examples return the point that is 1.5 standard deviations below the mean, although the actual values returned are different because the examples have different means. There is also a small rounding error in the last digit of the fifth example.

The last four examples all generate a message in the log, and return a missing value. The sixth example has the wrong number of arguments, the seventh example incorrectly specifies *sigma* but not *mu*, the eighth example specifies an invalid value for *sigma* and the ninth example specifies an invalid value for *q*, the cumulative density.

PROBNORM

Returns the cumulative density function at a given point for the standard Normal distribution with mean 0 and standard deviation 1. This function is similar to CDF – NORMAL and CDF – GAUSSIAN, with the mean and standard deviation set to the default values.

⇒ **PROBNORM** (**x**) ⇒

The calculated value is

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{1}{2}t^2\right) dt$$

Return type: Numeric

The return value is between 0 and 1 inclusive.

x**Type:** Numeric

The point at which to calculate the cumulative density.

Example

In this example, PROBNORM is used to calculate the value of the cumulative density function for the standard Normal distribution for various values of x. For comparison, this example also includes calls to the similar function CDF – NORMAL with mean 0 and standard deviation 1. The results are written to the log.

```
DATA _NULL_;

S1_PN = PROBNORM(0);
S1_CDF = CDF("NORMAL", 0, 0, 1);
PUT S1_PN=; PUT S1_CDF=; PUT;

S2_PN = PROBNORM(-1);
S2_CDF = CDF("NORMAL", -1, 0, 1);
PUT S2_PN=; PUT S2_CDF=; PUT;

S3_PN = PROBNORM(2);
S3_CDF = CDF("NORMAL", 2, 0, 1);
PUT S3_PN=; PUT S3_CDF=; PUT;

S4_PN = PROBNORM(50);
S4_CDF = CDF("NORMAL", 50, 0, 1);
PUT S4_PN=; PUT S4_CDF=; PUT;

RUN;
```

This produces the following output:

```
S1_PN=0.5
S1_CDF=0.5

S2_PN=0.1586552539
S2_CDF=0.1586552539

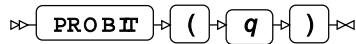
S3_PN=0.9772498681
S3_CDF=0.9772498681

S4_PN=1
S4_CDF=1
```

Each call to PROBNORM(x) returns the value of the cumulative density function at the specified point for a distribution with mean 0 and standard deviation 1. Each call to CDF("NORMAL", x, 0, 1) returns the same result as the equivalent call to PROBNORM(x).

PROBIT

Returns the quantile function at a given point for the standard Normal distribution with mean 0 and standard deviation 1. This function is similar to QUANTILE – NORMAL and QUANTILE – GAUSSIAN with mean and standard deviation set to the default values.



This function is the inverse of the PROBNORM function, which returns the cumulative density function for the standard Normal distribution with mean 0 and standard deviation 1.

This function is defined for $0 \leq q \leq 1$.

The calculated value is

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ (*infinium*) is the greatest lower bound of x , and $\text{CDF}(x)$ refers to the cumulative density function of the standard Normal distribution.

Note:

Function PROBIT requires a probability value, q , that satisfies $0 \leq q \leq 1$, whereas QUANTILE – NORMAL requires that q satisfies the stricter condition, $0 < q < 1$.

Return type: Numeric

If the wrong number of arguments are supplied, a missing value is returned.

q

Type: Numeric

The probability value for which to calculate the quantile.

Restriction: $0.0 \leq q \leq 1.0$

Example

In this example, the PROBIT function is used to calculate the value of the quantile function for the standard Normal distribution for various values of q . For comparison, this example includes calls to the similar function QUANTILE – NORMAL with mean 0 and standard deviation 1. This example also demonstrates the relationship with the two inverse functions, PROBNORM, and CDF – NORMAL.

The results are written to the log.

```

DATA _NULL_;

s1_pb = PROBIT(0.5);
s1_q = QUANTILE("NORMAL", 0.5, 0, 1);
s1_pn = PROBNORM(0);
s1_cdf = CDF("NORMAL", 0, 0, 1);
PUT s1_pb=; PUT s1_q=; PUT s1_pn=; PUT s1_cdf=; PUT ;
  
```

```

s2_pb = PROBIT(0.1586552539);
s2_q = QUANTILE("NORMAL", 0.1586552539, 0, 1);
s2_pn = PROBNORM(-1);
s2_cdf = CDF("NORMAL", -1, 0, 1);
PUT s2_pb=; PUT s2_q=; PUT s2_pn=; PUT s2_cdf=; PUT;

s3_pb = PROBIT(0.9772498681);
s3_q = QUANTILE("NORMAL", 0.9772498681, 0, 1);
s3_pn = PROBNORM(2);
s3_cdf = CDF("NORMAL", 2, 0, 1);
PUT s3_pb=; PUT s3_q=; PUT s3_pn=; PUT s3_cdf=; PUT;

s4_pb = PROBIT(1);
s4_q = QUANTILE("NORMAL", 1, 0, 1);
s4_pn = PROBNORM(1.797693E308);
s4_cdf = CDF("NORMAL", 1.797693E308, 0, 1);
PUT s4_pb=; PUT s4_q=; PUT s4_pn=; PUT s4_cdf=; PUT;

RUN;

```

This produces the following output:

```

s1_pb=0
s1_q=0
s1_pn=0.5
s1_cdf=0.5

s2_pb=-1
s2_q=-1
s2_pn=0.1586552539
s2_cdf=0.1586552539

s3_pb=2.0000000009
s3_q=2.0000000009
s3_pn=0.9772498681
s3_cdf=0.9772498681

s4_pb=1.797693E308
s4_q=.
s4_pn=1
s4_cdf=1

```

Each call to `PROBIT (q)` returns the value of the quantile function at point q for a distribution with mean 0 and standard deviation 1.

In most cases, the call to `QUANTILE ("NORMAL" , q , 0 , 1)` returns the same result as the call to `PROBIT (q)`. But the upper limit of q for the `PROBIT` function is 1, whereas the upper limit of q for the `QUANTILE` function is strictly less than 1. So `PROBIT (1)` returns a large positive value in `s4_pb`, but `QUANTILE ("NORMAL" , 1 , 0 , 1)` returns a missing value in `s4_q`.

Each call to `PROBNORM (x)` sets x to the value returned from `PROBIT (q)`. Since the value returned from `PROBIT (q)` is the point, x , where the value of the cumulative density function is q , the value returned from `PROBNORM (x)` is q , the cumulative density at point x . This is the value that was originally passed to `PROBIT`.

Each call to `CDF ("NORMAL" , x , 0 , 1)` returns the same result as the call to `PROBNORM (x)`.

DEVIANCE – NORMAL

Returns the deviance of the Normal distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – GAUSSIAN.

DEVIANCE ("NORMAL", x , μ)

Keyword **NORMAL** is an alias of **GAUSSIAN**, see *DEVIANCE – GAUSSIAN* [↗](#) (page 964).

Calculates the deviance, or goodness of fit, for the generalised linear model of the Normal distribution at point x based on the distribution mean μ (μ).

$$(x - \mu)^2$$

Return type: Numeric

If the wrong number of arguments are supplied, a missing value is returned.

x

Type: Numeric

The point at which to calculate the deviance.

If the argument contains a missing value, a missing value is returned.

μ

Type: Numeric

The distribution mean.

If the argument contains a missing value, a missing value is returned.

RAND – NORMAL

Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is similar to RANNOR, NORMAL and CALL RANNOR.

RAND ("NORMAL", mean , variance)

Each time you execute this function within a **DATA** step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

mean

Optional argument

Type: Numeric

The mean of the distribution.

variance

Optional argument

Type: Numeric

The variance of the distribution from the mean.

Example

In this example, a random number from the Normal distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("NORMAL", 10,5);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-1.746420844  
15.469459744  
11.235358351  
10.304586073  
2.0368974722
```

Running the `DATA` step again produces the following output.

```
The random numbers are:  
0.3816704915  
0.3473201428  
0.372621443  
0.3732903865  
0.4076553479
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(111);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("NORMAL", 10,5);  
    PUT result;  
  END;  
RUN;
```

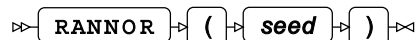
This produces the following output:

```
The random numbers are:  
1.3234328459  
12.179231016  
12.846849506  
16.768044651  
12.22727412
```

Running the `DATA` step again produces the same output.

RANNOR

Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is an alias of `NORMAL`. This function is similar to `RAND – NORMAL` and `CALL RANNOR`.



```
➤ [RANNOR] ➤ ( [seed] ) ➤
```

The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, a random number from the Normal distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANNOR(50);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.2772581053  
-0.059269116  
1.7234339317  
-0.520585025  
1.0438066355
```

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANNOR(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

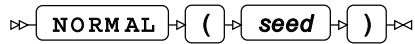
```
The random numbers are:  
1.4764823646  
-0.011249625  
1.8130791752  
0.5788490731  
0.0289116327
```

Running the DATA step again produces the following output.

```
The random numbers are:  
2.1501094753  
0.2106955146  
1.1363975832  
-1.689730536  
-0.054892306
```

NORMAL

Returns a random number from the Normal distribution based on the mean and variance of the distribution. This function is an alias of RANNOR. This function is similar to RAND – NORMAL and CALL RANNOR.



The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, a random number from the Normal distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = NORMAL(50);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.2772581053
-0.059269116
1.7234339317
-0.520585025
1.0438066355
```

If the initial seed is set to zero, each run of the `DATA` step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = NORMAL(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-19.05625376  
1.2717209805  
5.1594876086  
-0.221519437  
0.7377019906
```

Running the `DATA` step again produces the following output.

```
The random numbers are:  
-1.344530874  
0.6393523543  
0.6405996383  
-0.616669531  
0.2737123402
```

CALL RANNOR

Returns a random number from the Normal distribution. This routine is similar to function `RAND – NORMAL`, `NORMAL` and `RANNOR`.

```
CALL RANNOR ( seed , x ) ;
```

The diagram shows the syntax of the `CALL RANNOR` statement. It starts with a double arrow pointing to a box labeled `CALL RANNOR`. This is followed by an opening parenthesis `(`, a box labeled `seed`, a comma `,`, a box labeled `x`, a closing parenthesis `)`, and a semicolon `;`. The final arrow points to the right.

The distribution is parameterised using a mean of 0 and a standard deviation of 1.

Important:

The argument `seed` must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a `DATA` step, the stream of random numbers is initialised with the specified `seed`. Every time you update the `seed`, the stream is re-initialised.

To generate the same sequence of random numbers each time the `DATA` step is executed, set `seed` to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same `DATA` step. To generate a different sequence of random numbers each time the `DATA` step is executed, set `seed` to a positive value greater than or equal to 1. If the value specified for `seed` is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Normal distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  DO i = 1 TO 5;  
    CALL RANNOR(50, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
-1.668967435  
0.1304784715  
-2.189174283  
-1.682275958  
-0.564639193
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    CALL RANNOR(0, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:
-1.111876851
0.1786419171
0.4742460052
-0.957077094
1.1507782067
```

Running the DATA step again produces the following output.

```
The random numbers are:
-2.668104217
0.2081139588
0.9526697143
0.5769901728
-1.289722773
```

Normal mixture distribution

Functions for the Normal mixture distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – NORMALMIX [↗](#)..... 1211

$$\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right)$$

Returns the value of the probability density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of PMF – NORMALMIX.

PMF – NORMALMIX [↗](#)..... 1214

$$\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right)$$

Returns the value of the probability mass function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of PDF – NORMALMIX.

LOGPDF – NORMALMIX [↗](#)..... 1216

$$\log \left[\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right) \right]$$

Returns the value of the natural logarithm of the probability density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of LOGPMF – NORMALMIX.

LOGPMF – NORMALMIX [↗](#)..... 1219

$$\log \left[\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right) \right]$$

Returns the value of the natural logarithm of the probability mass function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of LOGPDF – NORMALMIX.

CDF – NORMALMIX [↗](#)..... 1222

$$\sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt$$

Returns the value of the cumulative density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.

LOGCDF – NORMALMIX [↗](#)..... 1225

$$\log \left[\sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt \right]$$

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.

SDF – NORMALMIX [↗](#)..... 1228

$$1 - \sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt$$

Returns the value of the survival function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.

LOGSDF – NORMALMIX [↗](#)..... 1231

$$\log \left[1 - \sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt \right]$$

Returns the value of the natural logarithm of the survival function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.

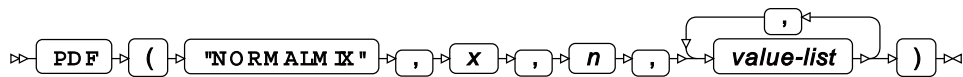
QUANTILE – NORMALMIX [↗](#)..... 1234

$$\inf \{x: q \leq \text{CDF}(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma})\}$$

Returns the value of the quantile function at a given point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.

PDF – NORMALMIX

Returns the value of the probability density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of `PMF – NORMALMIX`.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right)$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

n**Type:** Numeric**Restriction:** Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list**Type:** NumericA list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, PDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the weighted sums of the values returned by PDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("NORMALMIX", 0, 1, 1, 2, 3);
s2 = PDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s1= s2=;

s3 = 1 * PDF("NORMAL", 0, 2, 3);
s4 = 0.3 * PDF("NORMAL", 1.5, 1, 4) + 0.7 * PDF("NORMAL", 1.5, -3, 0.25);
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=0.1064826685 s2=0.0296878265
s3=0.1064826685 s4=0.0296878265
```

Variable `s1` contains the value at $x = 0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x = 1.5$ for a Normal mixture distribution consisting of two components ($n = 2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables *s3* and *s4* contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable *s3* uses the value at $x = 0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable *s4* uses the sum of:

- the value at $x = 1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x = 1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, *s1* and *s3* are identical and *s2* and *s4* are identical.

Argument errors

In this example, PDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = PDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = PDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = PDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = PDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = PDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

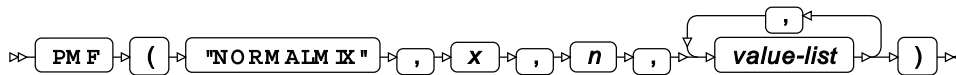
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n = 2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n = 0$ and $n = -1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

PMF – NORMALMIX

Returns the value of the probability mass function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of PDF – NORMALMIX.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right)$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, PMF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the weighted sums of the values returned by PMF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("NORMALMIX", 0, 1, 1, 2, 3);
s2 = PMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s1= s2=;

s3 = 1 * PMF("NORMAL", 0, 2, 3);
s4 = 0.3 * PMF("NORMAL", 1.5, 1, 4) + 0.7 * PMF("NORMAL", 1.5, -3, 0.25);
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=0.1064826685 s2=0.0296878265
s3=0.1064826685 s4=0.0296878265
```

Variable `s1` contains the value at $x = 0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x = 1.5$ for a Normal mixture distribution consisting of two components ($n = 2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x = 0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x = 1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x = 1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, $s1$ and $s3$ are identical and $s2$ and $s4$ are identical.

Argument errors

In this example, PMF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = PMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = PMF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = PMF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = PMF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = PMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

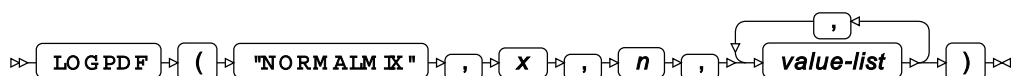
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n = 2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n = 0$ and $n = -1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

LOGPDF – NORMALMIX

Returns the value of the natural logarithm of the probability density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of LOGPMF – NORMALMIX.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument *n* specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \log \left[\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right) \right]$$

where

- *n* is the number of components in the mixture
- **W** is a vector containing the weights to associate with each component in the mixture
- **μ** is a vector containing the means of the components in the mixture
- **σ** is a vector containing the standard deviations of the components in the mixture
- *w_i* is the weight that the *i*th Normal distribution contributes to the mixture
- *μ_i* is the mean of the *i*th Normal distribution in the mixture
- *σ_i* is the standard deviation of the *i*th Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- *n* values specifying the weights to apply to each of the *n* Normal distributions in the mixture

- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, LOGPDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the natural logarithm of the weighted sums of the values returned by PDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("NORMALMIX", 0, 1, 1, 2, 3);
s2 = LOGPDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s1= s2=;

s3 = LOG (PDF("NORMAL", 0, 2, 3));
s4 = LOG (0.3 * PDF("NORMAL", 1.5, 1, 4) + 0.7 * PDF("NORMAL", 1.5, -3, 0.25));
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=-2.239773044 s2=-3.517018199
s3=-2.239773044 s4=-3.517018199
```

Variable `s1` contains the value at $x = 0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x = 1.5$ for a Normal mixture distribution consisting of two components ($n = 2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x = 0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x = 1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x = 1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, LOGPDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = LOGPDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = LOGPDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = LOGPDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = LOGPDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = LOGPDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

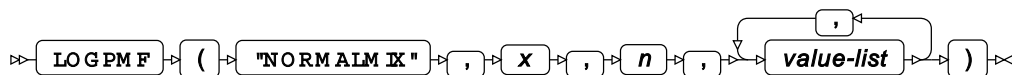
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

LOGPMF – NORMALMIX

Returns the value of the natural logarithm of the probability mass function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations. This function is an alias of LOGPDF – NORMALMIX.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \log \left[\sum_{i=1}^n \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right) \right) \right]$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, LOGPMF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the natural logarithm of the weighted sums of the values returned by PMF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGPMF("NORMALMIX", 0, 1, 1, 2, 3);
  s2 = LOGPMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
  PUT s1= s2=;

  s3 = LOG (PMF("NORMAL", 0, 2, 3));
  s4 = LOG (0.3 * PMF("NORMAL", 1.5, 1, 4) + 0.7 * PMF("NORMAL", 1.5, -3, 0.25));
  PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=-2.239773044 s2=-3.517018199
s3=-2.239773044 s4=-3.517018199
```

Variable `s1` contains the value at $x=0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x=1.5$ for a Normal mixture distribution consisting of two components ($n=2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x=0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x=1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x=1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, LOGPMF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = LOGPMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = LOGPMF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = LOGPMF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = LOGPMF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = LOGPMF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

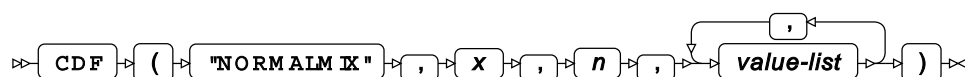
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

CDF – NORMALMIX

Returns the value of the cumulative density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp\left(-\frac{1}{2}\left(\frac{t-\mu_i}{\sigma_i}\right)^2\right) \right) dt$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, CDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the weighted sums of the values returned by CDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("NORMALMIX", 0, 1, 1, 2, 3);
s2 = CDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s1= s2=;

s3 = CDF("NORMAL", 0, 2, 3);
s4 = 0.3 * CDF("NORMAL", 1.5, 1, 4) + 0.7 * CDF("NORMAL", 1.5, -3, 0.25);
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=0.2524925375 s2=0.8649214674
s3=0.2524925375 s4=0.8649214674
```

Variable `s1` contains the value at $x = 0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x = 1.5$ for a Normal mixture distribution consisting of two components ($n = 2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x = 0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x = 1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x = 1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, CDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = CDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = CDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = CDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = CDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = CDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

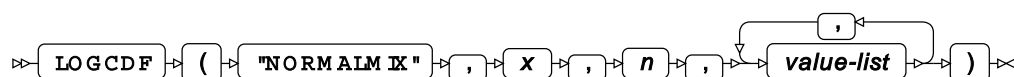
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

LOGCDF – NORMALMIX

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \log \left[\sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt \right]$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, LOGCDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the natural logarithm of the weighted sums of the values returned by CDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF("NORMALMIX", 0, 1, 1, 2, 3);
s2 = LOGCDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s1= s2=;

s3 = LOG (CDF("NORMAL", 0, 2, 3));
s4 = LOG (0.3 * CDF("NORMAL", 1.5, 1, 4) + 0.7 * CDF("NORMAL", 1.5, -3, 0.25));
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=-1.376373585 s2=-0.145116565
s3=-1.376373585 s4=-0.145116565
```

Variable `s1` contains the value at $x=0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x=1.5$ for a Normal mixture distribution consisting of two components ($n=2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x=0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x=1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x=1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, LOGCDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = LOGCDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = LOGCDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = LOGCDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = LOGCDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = LOGCDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

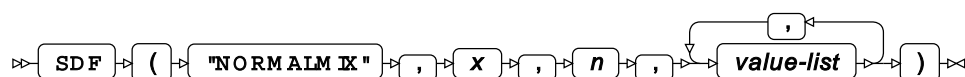
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

SDF – NORMALMIX

Returns the value of the survival function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = 1 - \sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp\left(-\frac{1}{2}\left(\frac{t - \mu_i}{\sigma_i}\right)^2\right) \right) dt$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the value of the survival function.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, SDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the weighted sums of the values returned by SDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = SDF("NORMALMIX", 0, 1, 1, 2, 3);
  s2 = SDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
  PUT s1= s2=;

  s3 = SDF("NORMAL", 0, 2, 3);
  s4 = 0.3 * SDF("NORMAL", 1.5, 1, 4) + 0.7 * SDF("NORMAL", 1.5, -3, 0.25);
  PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=0.7475074625 s2=0.1350785326
s3=0.7475074625 s4=0.1350785326
```

Variable `s1` contains the value at $x = 0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x = 1.5$ for a Normal mixture distribution consisting of two components ($n = 2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x = 0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x = 1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x = 1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, SDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = SDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = SDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = SDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = SDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = SDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

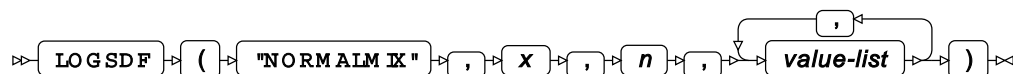
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

LOGSDF – NORMALMIX

Returns the value of the natural logarithm of the survival function at a specified point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value for the Normal mixture distribution is

$$f(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \log \left[1 - \sum_{i=1}^n \int_{-\infty}^x \left(\frac{w_i}{\sqrt{2\pi} \sigma_i} \exp \left(-\frac{1}{2} \left(\frac{t - \mu_i}{\sigma_i} \right)^2 \right) \right) dt \right]$$

where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture
- w_i is the weight that the i^{th} Normal distribution contributes to the mixture
- μ_i is the mean of the i^{th} Normal distribution in the mixture
- σ_i is the standard deviation of the i^{th} Normal distribution in the mixture
- $\sum_{i=1}^n w_i = 1$
- $\sigma_i > 0$ for $1 \leq i \leq n$.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, LOGSDF – NORMALMIX is called for various mixtures of Normal distributions. The results are compared with the values calculated from the natural logarithm of the weighted sums of the values returned by SDF – NORMAL for the individual Normal distributions. The results are written to the log.

```
DATA _NULL_;

  s1 = LOGSDF("NORMALMIX", 0, 1, 1, 2, 3);
  s2 = LOGSDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25);
  PUT s1= s2=;

  s3 = LOG (SDF("NORMAL", 0, 2, 3));
  s4 = LOG (0.3 * SDF("NORMAL", 1.5, 1, 4) + 0.7 * SDF("NORMAL", 1.5, -3, 0.25));
  PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=-0.291010991 s2=-2.001898947
s3=-0.291010991 s4=-2.001898947
```

Variable `s1` contains the value at $x=0$ for a Normal mixture distribution consisting of a single component with unit weighting, a mean of 2 and a standard deviation of 3.

Variable `s2` contains the value at $x=1.5$ for a Normal mixture distribution consisting of two components ($n=2$) with weightings of 0.3 and 0.7, means of 1 and -3 and standard deviations of 4 and 0.25.

For comparison, variables `s3` and `s4` contain the values for the same two mixtures of Normal distributions as above. Here, the values are calculated directly using weighted sums of the equivalent Normal distributions. Variable `s3` uses the value at $x=0$ for a single Normal distribution with a mean of 2 and a standard deviation of 3. Variable `s4` uses the sum of:

- the value at $x=1.5$ for a Normal distribution with a mean of 1 and a standard deviation of 4, multiplied by a weighting factor of 0.3
- the value at $x=1.5$ for a Normal distribution with a mean of -3 and a standard deviation of 0.25, multiplied by a weighting factor of 0.7

In this example, `s1` and `s3` are identical and `s2` and `s4` are identical.

Argument errors

In this example, LOGSDF – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGSDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = LOGSDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = LOGSDF("NORMALMIX", 1.5, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = LOGSDF("NORMALMIX", 1.5, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = LOGSDF("NORMALMIX", 1.5, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;

s6 = LOGSDF("NORMALMIX", 1.5, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

RUN;
```

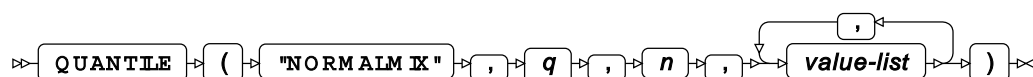
All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

QUANTILE – NORMALMIX

Returns the value of the quantile function at a given point for the Normal mixture distribution with the specified mixture proportions, means and standard deviations.



The Normal mixture distribution is a mixture distribution that is derived as the sum of multiple distinct Normal distributions, each with a specified mean and standard deviation, combined in specified proportions.

Argument n specifies the number of Normal distributions in the mixture and argument *value-list* specifies the weights, means and standard deviations of the individual distributions in the mixture.

The calculated value, x , for the Normal mixture distribution is

$$f(q; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \inf \{x : q \leq \text{CDF}(x; n, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma})\}$$

where $x \in \mathbb{R}$, $\inf\{x\}$ (*infimum*) is the greatest lower bound of x , and $\text{CDF}(x; N, \mathbf{W}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ is the cumulative density function of the Normal mixture distribution where

- n is the number of components in the mixture
- \mathbf{W} is a vector containing the weights to associate with each component in the mixture
- $\boldsymbol{\mu}$ is a vector containing the means of the components in the mixture
- $\boldsymbol{\sigma}$ is a vector containing the standard deviations of the components in the mixture

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

n

Type: Numeric

Restriction: Must be a positive integer.

The number of components in the mixture.

If the argument is out of range, a missing value is returned.

value-list

Type: Numeric

A list of $3n$ numbers specifying the composition of the mixture, and containing, in this order:

- n values specifying the weights to apply to each of the n Normal distributions in the mixture
- n values specifying the means of each of the n Normal distributions in the mixture
- n values specifying the standard deviations of each of the n Normal distributions in the mixture

The list must contain exactly $3n$ items; otherwise a missing value is returned.

The sum of the weights must be exactly 1; otherwise a missing value is returned.

Each standard deviation must be greater than 0 (zero); otherwise a missing value is returned.

Basic example

In this example, QUANTILE – NORMALMIX is called for different mixtures of Normal distributions. The returned values are then passed to CDF – NORMALMIX for the same mixture distributions. The results are written to the log.

```
DATA _NULL_;

s1 = QUANTILE("NORMALMIX", 0.75, 2, 0.3, 0.7, 1, -3, 4, 0.25);
s2 = QUANTILE("NORMALMIX", 0.95, 3, 0.2, 0.3, 0.5, 0, 1, 2, 1, 2, 3);
PUT s1= s2=;

s3 = CDF("NORMALMIX", s1, 2, 0.3, 0.7, 1, -3, 4, 0.25);
s4 = CDF("NORMALMIX", s2, 3, 0.2, 0.3, 0.5, 0, 1, 2, 1, 2, 3);
PUT s3= s4=;

RUN;
```

This produces the following output:

```
s1=-2.437161099 s2=5.9172964047
s3=0.75 s4=0.95
```

Variable *s1*=-2.437161099 is the value returned by QUANTILE – NORMALMIX for $q=0.75$ for a Normal mixture consisting of two components ($N=2$). The mixture has weightings 0.3 and 0.7, means 1 and -3 and standard deviations 4 and 0.25.

Variable *s2* is the value returned by QUANTILE – NORMALMIX for $q=0.95$ for a Normal mixture consisting of three components ($N=3$). The mixture has weightings 0.2, 0.3 and 0.5, means 0, 1 and -2 and standard deviations 1, 2 and 3.

As a comparison, variables *s3* and *s4* contain the results of passing *s1* and *s2* to CDF – NORMALMIX. As QUANTILE – NORMALMIX and CDF – NORMALMIX are inverse functions, the values returned from CDF – NORMALMIX are the same as the values originally passed to QUANTILE – NORMALMIX.

Argument errors

In this example, QUANTILE – NORMALMIX is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = QUANTILE("NORMALMIX", 0.8649214674, 2, 0.3, 0.7, 1, -3, 4, 0.25, 1);
PUT s1=;

s2 = QUANTILE("NORMALMIX", 0.8649214674, 2, 0.3, 0.7, 1, -3, 4);
PUT s2=;

s3 = QUANTILE("NORMALMIX", 0.8649214674, 0, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s3=;
s4 = QUANTILE("NORMALMIX", 0.8649214674, -1, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s4=;

s5 = QUANTILE("NORMALMIX", 0.8649214674, 2, 0.3, 0.8, 1, -3, 4, 0.25);
PUT s5=;
```

```
s6 = QUANTILE("NORMALMIX", 0.8649214674, 2, 0.3, 0.7, 1, -3, -4, 0.25);
PUT s6=;

s7 = QUANTILE("NORMALMIX", 0, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s7;

s8 = QUANTILE("NORMALMIX", -1, 2, 0.3, 0.7, 1, -3, 4, 0.25);
PUT s8=;

RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example specifies a Normal mixture distribution with two components ($n=2$), then provides seven more arguments instead of six. The second example also specifies a Normal mixture distribution with two components, but only provides five more arguments instead of six.

The third and fourth examples specify invalid values for n ($n=0$ and $n=-1$ respectively). The fifth example has weights that don't sum to unity (0.3 and 0.8), and the sixth example has an invalid standard deviation of -4.

The seventh and eighth examples specify invalid values for q ($q=0$ and $q=0$ respectively).

Pareto distribution

Functions for the Pareto distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

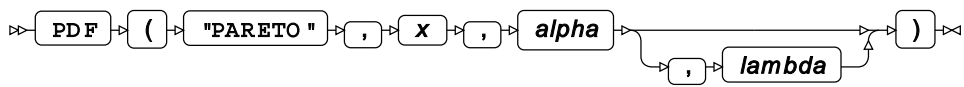
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – PARETO ↗	1238
Returns the probability density of the Pareto distribution. This function is an alias of PMF – PARETO.	
PMF – PARETO ↗	1239
Returns the probability mass of the Pareto distribution. This function is an alias of PDF – PARETO.	
LOGPDF – PARETO ↗	1239
Returns the natural logarithm of the probability density of the Pareto distribution. This function is an alias of LOGPMF – PARETO.	
LOGPMF – PARETO ↗	1240
Returns the natural logarithm of the probability mass of the Pareto distribution. This function is an alias of LOGPDF – PARETO.	

CDF – PARETO ↗	1240
Returns the cumulative density of the Pareto distribution.	
LOGCDF – PARETO ↗	1241
Returns the natural logarithm of the cumulative density of the Pareto distribution.	
SDF – PARETO ↗	1241
Returns the survival of the Pareto distribution.	
LOGSDF – PARETO ↗	1242
Returns the natural logarithm of the survival of the Pareto distribution.	
QUANTILE – PARETO ↗	1243
Returns the quantile of the Pareto distribution.	
RAND – PARETO ↗	1243
Returns a random number from the Pareto distribution based on the shape.	

PDF – PARETO

Returns the probability density of the Pareto distribution. This function is an alias of PMF – PARETO.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

alpha

Type: Numeric

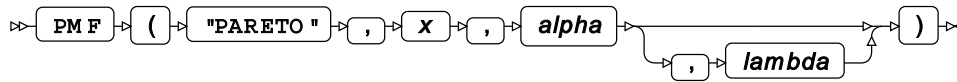
lambda

Optional argument

Type: Numeric

PMF – PARETO

Returns the probability mass of the Pareto distribution. This function is an alias of PDF – PARETO.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

alpha

Type: Numeric

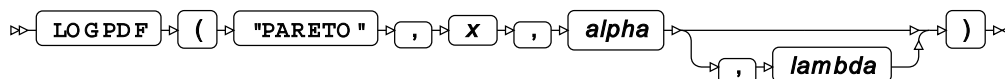
lambda

Optional argument

Type: Numeric

LOGPDF – PARETO

Returns the natural logarithm of the probability density of the Pareto distribution. This function is an alias of LOGPMF – PARETO.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

alpha

Type: Numeric

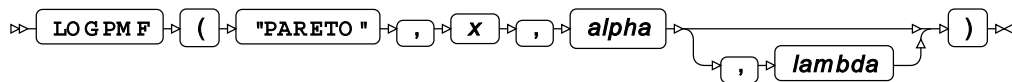
lambda

Optional argument

Type: Numeric

LOGPMF – PARETO

Returns the natural logarithm of the probability mass of the Pareto distribution. This function is an alias of LOGPDF – PARETO.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

alpha

Type: Numeric

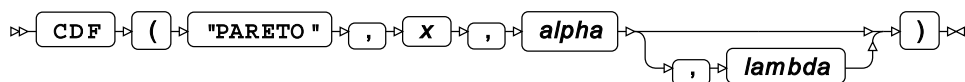
lambda

Optional argument

Type: Numeric

CDF – PARETO

Returns the cumulative density of the Pareto distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

alpha

Type: Numeric

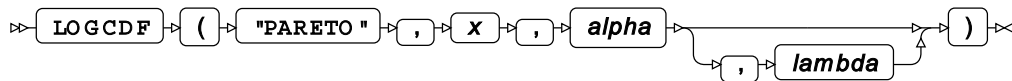
lambda

Optional argument

Type: Numeric

LOGCDF – PARETO

Returns the natural logarithm of the cumulative density of the Pareto distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

alpha

Type: Numeric

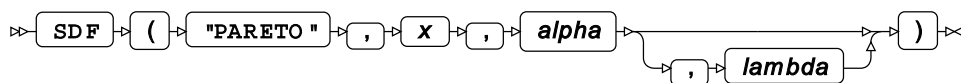
lambda

Optional argument

Type: Numeric

SDF – PARETO

Returns the survival of the Pareto distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

alpha

Type: Numeric

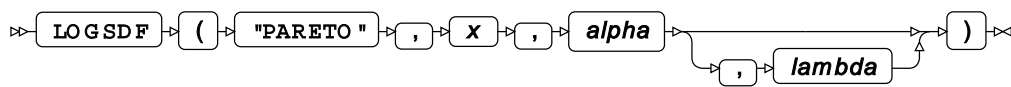
lambda

Optional argument

Type: Numeric

LOGSDF – PARETO

Returns the natural logarithm of the survival of the Pareto distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

alpha

Type: Numeric

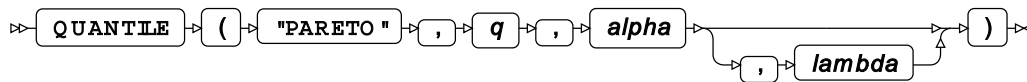
lambda

Optional argument

Type: Numeric

QUANTILE – PARETO

Returns the quantile of the Pareto distribution.



Return type: Numeric

q

Type: Numeric

alpha

Type: Numeric

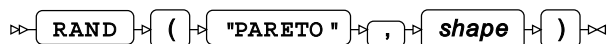
lambda

Optional argument

Type: Numeric

RAND – PARETO

Returns a random number from the Pareto distribution based on the shape.



This function does not take any variable arguments.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

shape**Type:** Numeric

The shape of the distribution.

Example

In this example, a random number from the Pareto distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("PARETO", 15);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
1.1368619062  
1.0627707892  
1.0061861723  
1.0720440322  
1.0517167879
```

Running the DATA step again produces the following output.

```
The random numbers are:  
1.1761579249  
1.0538461792  
1.0696985308  
1.0637153245  
1.0401712807
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  CALL STREAMINIT(12);  
  DO i = 1 TO 5;  
    result = RAND("PARETO", 15);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
1.0365897645  
1.0258496779  
1.0765356555  
1.0116339965  
1.0525305894
```

Running the DATA step again produces the same output.

Poisson distribution

Functions and CALL routines for the Poisson distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – POISSON [↗](#)..... 1247

$$\frac{e^{-\lambda} \lambda^n}{n!}$$

Returns the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of PMF – POISSON.

PMF – POISSON [↗](#)..... 1249

$$\frac{e^{-\lambda} \lambda^n}{n!}$$

Returns the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of PDF – POISSON.

LOGPDF – POISSON [↗](#)..... 1251

$$\log\left(\frac{e^{-\lambda} \lambda^n}{n!}\right)$$

Returns the natural logarithm of the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of LOGPMF – POISSON.

LOGPMF – POISSON [↗](#)..... 1253

$$\log\left(\frac{e^{-\lambda} \lambda^n}{n!}\right)$$

Returns the natural logarithm of the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of LOGPDF – POISSON.

CDF – POISSON [↗](#)..... 1255

$$\sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Returns the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence. This function is similar to POISSON.

POISSON [↗](#)..... 1257

$$\sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Returns the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence. This function is similar to CDF – POISSON.

LOGCDF – POISSON [↗](#).....1259

$$\log \left(\sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!} \right)$$

Returns the natural logarithm of the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence.

SDF – POISSON [↗](#)..... 1261

$$1 - \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Returns the survival density for a Poisson distribution for a specified minimum number of events and mean rate of occurrence.

LOGSDF – POISSON [↗](#).....1263

$$\log \left(1 - \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!} \right)$$

Returns the natural logarithm of the survival density for a Poisson distribution, for a specified minimum number of events and mean rate of occurrence.

QUANTILE – POISSON [↗](#).....1265

$$f(q) = \inf \{x : q \leq \text{CDF}(x)\}$$

Returns the quantile for a Poisson distribution for a given cumulative density and mean rate of occurrence.

DEVIANCE – POISSON [↗](#)..... 1267

$$\begin{cases} \text{if } x=0 & \text{return } 2\mu \\ \text{otherwise} & \text{return } 2(x \log \frac{x}{\mu} - (x-\mu)) \end{cases}$$

Returns the deviance of the Poisson distribution at a specified point, based on the distribution mean.

RAND – POISSON [↗](#)..... 1269

Returns a random number from the Poisson distribution based on the mean. This function is similar to RANPOI and CALL RANPOI.

RANPOI [↗](#).....1271

Returns a random number from the Poisson distribution based on the mean. This function is similar to RAND – POISSON and CALL RANPOI.

CALL RANPOI [↗](#)..... 1272

Returns a random number from the Poisson distribution based on the mean. This routine is similar to function RAND – POISSON and RANPOI.

PDF – POISSON

Returns the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of PMF – POISSON.



The probability density function (PDF) for the Poisson distribution gives the probability of a specified number of events n occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The probability density function is defined as:

$$f(n; \lambda) = \frac{e^{-\lambda} \lambda^n}{n!}$$

Return type: Numeric

n

Type: Numeric

The number of events for which to return the probability mass. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1 = PDF("POISSON", 1, 1);
PUT S1=;
S2 = PDF("POISSON", 0, 1);
PUT S2=;
RUN;
```

This produces the following output:

```
S1=0.3678794412
S2=0.3678794412
```

The first example returns the probability mass for 1 event occurring if the rate of occurrence is 1. The second example repeats this for 0 events, showing that the result is the same.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = PDF("POISSON", 1, 3);  
PUT S3=;  
S4 = PDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=0.1493612051  
S4=0.0148725131
```

These two examples show the effect of an increasing rate of occurrence on probability mass.

Example – number of events

```
DATA _NULL_;  
S5 = PDF("POISSON", 3, 1);  
PUT S5=;  
S6 = PDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.0613132402  
S6=0.0005109437
```

These two examples show the effect of an increasing number of events on probability mass.

Example – decimal values

```
DATA _NULL_;  
S7 = PDF("POISSON", 1, 0.4);  
PUT S7=;  
S8 = PDF("POISSON", 1.1, 1);  
PUT S8=;  
RUN;
```

This produces the following output:

```
S7=0.2681280184  
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

PMF – POISSON

Returns the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of PDF – POISSON.



The probability mass function (PMF) for a Poisson distribution gives the probability of a specified number of events n occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The probability mass function is defined as:

$$f(n; \lambda) = \frac{e^{-\lambda} \lambda^n}{n!}$$

Return type: Numeric

n

Type: Numeric

The number of events for which to return the probability mass. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1 = PMF("POISSON", 1, 1);  
PUT S1=;  
S2 = PMF("POISSON", 0, 1);  
PUT S2=;  
RUN;
```

This produces the following output:

```
S1=0.3678794412  
S2=0.3678794412
```

The first example gives the probability mass for 1 event occurring if the rate of occurrence is 1, whilst the second example repeats this for 0 events, showing that the result is the same.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = PMF("POISSON", 1, 3);  
PUT S3=;  
S4 = PMF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=0.1493612051  
S4=0.0148725131
```

These two examples show the effect of an increasing rate of occurrence on probability mass.

Example – number of events

```
DATA _NULL_;  
S5 = PMF("POISSON", 3, 1);  
PUT S5=;  
S6 = PMF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.0613132402  
S6=0.0005109437
```

These two examples show the effect of an increasing number of events on probability mass.

Example – decimal values

```
DATA _NULL_;  
S7 = PMF("POISSON", 1, 0.4);  
PUT S7=;  
S8 = PMF("POISSON", 1.1, 1);  
PUT S8=;  
RUN;
```

This produces the following output:

```
S7=0.2681280184  
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

LOGPDF – POISSON

Returns the natural logarithm of the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of LOGPMF – POISSON.

LOGPDF ("POISSON" , *n* , *lambda*)

The natural logarithm of the probability density function (LOGPDF) for a Poisson distribution gives the natural logarithm of the probability of a specified number of events *n* occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The natural logarithm of the probability density function is defined as:

$$f(n; \lambda) = \log \left(\frac{e^{-\lambda} \lambda^n}{n!} \right)$$

Return type: Numeric

n

Type: Numeric

The number of events for which to return the natural logarithm of the probability mass. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1 = LOGPDF("POISSON", 1, 1);  
PUT S1=;  
S2 = LOGPDF("POISSON", 0, 1);  
PUT S2=;  
RUN;
```

This produces the following output:

```
S1=-1  
S2=-1
```

The first example gives the natural logarithm of the probability mass for 1 event occurring if the rate of occurrence is 1. The second example repeats this for 0 events, showing that the result is the same.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = LOGPDF("POISSON", 1, 3);  
PUT S3=;  
S4 = LOGPDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=-1.901387711  
S4=-4.208240531
```

These two examples show the effect of an increasing rate of occurrence on the natural logarithm of the probability mass.

Example – number of events

```
DATA _NULL_;  
S5 = LOGPDF("POISSON", 3, 1);  
PUT S5=;  
S6 = LOGPDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=-2.791759469  
S6=-7.579251212
```

These two examples show the effect of an increasing number of events on the natural logarithm of the probability mass.

Example – decimal values

```
DATA _NULL_;  
S7 = LOGPDF("POISSON", 1, 0.4);  
PUT S7=;  
S8 = LOGPDF("POISSON", 1.1, 1);  
PUT S8=;  
RUN;
```

This produces the following output:

```
S7=-1.316290732  
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

LOGPMF – POISSON

Returns the natural logarithm of the probability mass for a Poisson distribution for a specified number of events and mean rate of occurrence. This function is an alias of LOGPDF – POISSON.

→ LOGPMF (("POISSON" , *n* , *lambda*) →

The natural logarithm of the probability mass function (LOGPMF) for a Poisson distribution gives the natural logarithm of the probability of a specified number of events *n* occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The natural logarithm of the probability mass function is defined as:

$$f(n; m) = \log\left(\frac{e^{-\lambda}\lambda^n}{n!}\right)$$

Return type: Numeric

n

Type: Numeric

The number of events for which to return the natural logarithm of the probability mass. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1 = LOGPMF("POISSON", 1, 1);
PUT S1=;
S2 = LOGPMF("POISSON", 0, 1);
PUT S2=;
RUN;
```

This produces the following output:

```
S1=-1
S2=-1
```

The first example gives the natural logarithm of the probability mass for 1 event occurring if the rate of occurrence is 1. The second example repeats this for 0 events, showing that the result is the same.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = LOGPMF("POISSON", 1, 3);  
PUT S3=;  
S4 = LOGPMF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=-1.901387711  
S4=-4.208240531
```

These two examples show the effect of an increasing rate of occurrence on the natural logarithm of the probability mass.

Example – number of events

```
DATA _NULL_;  
S5 = LOGPMF("POISSON", 3, 1);  
PUT S5=;  
S6 = LOGPMF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=-2.791759469  
S6=-7.579251212
```

These two examples show the effect of an increasing number of events on the natural logarithm of the probability mass.

Example – decimal values

```
DATA _NULL_;  
S7 = LOGPMF("POISSON", 1, 0.4);  
PUT S7=;  
S8 = LOGPMF("POISSON", 1.1, 1);  
PUT S8=;  
RUN;
```

This produces the following output:

```
S7=-1.316290732  
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

CDF – POISSON

Returns the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence. This function is similar to POISSON.

⇒ CDF ("POISSON" , *n* , *lambda*) ⇒

The cumulative density function (CDF) for a Poisson distribution gives the probability of occurrence of up to and including a specified maximum number of events *n* in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The cumulative density function is defined as:

$$f(n; m) = \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Return type: Numeric

n

Type: Numeric

The upper limit of events for which to return the cumulative density. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1 = CDF("POISSON", 1, 1);
PUT S1=;
S2 = CDF("POISSON", 1, 3);
PUT S2=;
RUN;
```

This produces the following output:

```
S1=0.7357588823
S2=0.1991482735
```

The first example gives the cumulative density for 1 event, which is equivalent to the probability of both 0 events and 1 event occurring. This is calculated with a rate of occurrence of 1. The second example triples the rate of occurrence of events, reducing the cumulative density.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = CDF("POISSON", 1, 3);  
PUT S3=;  
S4 = CDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=0.1991482735  
S4=0.0173512652
```

These two examples show the effect of an increasing rate of occurrence on cumulative density, given the same number of events.

Example – number of events

```
DATA _NULL_;  
S5 = CDF("POISSON", 3, 1);  
PUT S5=;  
S6 = CDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.9810118431  
S6=0.9999167589
```

These two examples show the effect of an increasing number of events on the cumulative density, given the same rate of occurrence.

Example – decimal values

```
DATA _NULL_;  
S7 = CDF("POISSON", 1, 0.4);  
PUT S7=;  
S8 = CDF("POISSON", 1.1, 0);  
PUT S8=;  
RUN;
```

This produces the following output:

```
S7=0.9384480644  
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

POISSON

Returns the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence. This function is similar to CDF – POISSON.

⇒ POISSON ((*lambda* , *n*)) ⇐

Note:

Function POISSON differs from CDF – POISSON in the order of its arguments.

The Poisson function gives the cumulative density for a Poisson distribution. The cumulative density is defined as the probability of occurrence of up to and including a specified maximum number of events *n* in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The cumulative density function is defined as:

$$f(n; \lambda) = \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Return type: Numeric

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

n

Type: Numeric

The upper limit of events for which to return the cumulative density. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: *n* ≥ 0 and must be integer

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1 = POISSON(1, 1);
PUT S1=;
S2 = POISSON(3, 1);
PUT S2=;
RUN;
```

This produces the following output:

```
S1=0.7357588823  
S2=0.1991482735
```

The first example gives the cumulative density for 1 event, which is equivalent to the probability of both 0 events and 1 event occurring. This is calculated with a rate of occurrence of 1. The second example triples the rate of occurrence of events, reducing the cumulative density.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = POISSON(3, 1);  
PUT S3=;  
S4 = POISSON(6, 1);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=0.1991482735  
S4=0.0173512652
```

These two examples show the effect of an increasing rate of occurrence on cumulative density, given the same number of events.

Example – number of events

```
DATA _NULL_;  
S5 = POISSON(1, 3);  
PUT S5=;  
S6 = POISSON(1, 6);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.9810118431  
S6=0.9999167589
```

These two examples show the effect of an increasing number of events on cumulative density, given the same rate of occurrence.

Example – decimal values

```
DATA _NULL_;  
S7 = POISSON(0.4, 1);  
PUT S7=;  
S8 = POISSON(0, 1.1);  
PUT S8=;  
RUN;
```


This produces the following output:

```
S7=0.9384480644
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

LOGCDF – POISSON

Returns the natural logarithm of the cumulative density for a Poisson distribution up to a specified maximum number of events and mean rate of occurrence.

```
LOGCDF ( "POISSON" , n , lambda )
```

The natural logarithm of the cumulative density function (LOGCDF) for a Poisson distribution gives the natural logarithm of the probability of occurrence of up to and including a specified maximum number of events n in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The natural logarithm of the cumulative density function is defined as:

$$f(n; \lambda) = \log \left(\sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!} \right)$$

Return type: Numeric

n

Type: Numeric

The upper limit of events for which to return the natural logarithm of the cumulative density. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1 = LOGCDF("POISSON", 1, 1);  
PUT S1=;  
S2 = LOGCDF("POISSON", 1, 3);  
PUT S2=;  
RUN;
```

This produces the following output:

```
S1=-0.306852819  
S2=-1.613705639
```

The first example gives the natural logarithm of the cumulative density for 1 event, which is equivalent to the natural logarithm of the probability of both 0 events and 1 event occurring. This is calculated with a rate of occurrence of 1. The second example triples the rate of occurrence of events, reducing the result.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = LOGCDF("POISSON", 1, 3);  
PUT S3=;  
S4 = LOGCDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=-1.613705639  
S4=-4.054089851
```

These two examples show the effect of an increasing rate of occurrence on the natural logarithm of the cumulative density, given the same number of events.

Example – number of events

```
DATA _NULL_;  
S5 = LOGCDF("POISSON", 3, 1);  
PUT S5=;  
S6 = LOGCDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=-0.019170747  
S6=-0.000083245
```

These two examples show the effect of an increasing number of events on the natural logarithm of the cumulative density, given the same rate of occurrence.

Example – decimal values

```
DATA _NULL_;
  S7 = LOGCDF("POISSON", 1, 0.4);
  PUT S7=;
  S8 = LOGCDF("POISSON", 1.1, 0);
  PUT S8=;
RUN;
```

This produces the following output:

```
S7=-0.063527763
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

SDF – POISSON

Returns the survival density for a Poisson distribution for a specified minimum number of events and mean rate of occurrence.

```
SDF ( "POISSON" , n , lambda )
```

The survival density function (SDF) for a Poisson distribution gives the probability of more than a specified number of events n occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The survival density function is defined as:

$$f(n; \lambda) = 1 - \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!}$$

Return type: Numeric

n

Type: Numeric

The start of the range of events for which to return the survival density. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *n*, the number of events.

Restriction: $\lambda > 0$

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;  
S1 = SDF("POISSON", 1, 1);  
PUT S1=;  
S2 = SDF("POISSON", 1, 3);  
PUT S2=;  
RUN;
```

This produces the following output:

```
S1=0.2642411177  
S2=0.8008517265
```

The first example above gives the survival density for 1 event, which is the probability of 2 or more events occurring. This is calculated with a rate of occurrence of 1. The second example triples the rate of occurrence of events, greatly increasing the survival density.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = SDF("POISSON", 1, 3);  
PUT S3=;  
S4 = SDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=0.8008517265  
S4=0.9826487348
```

These two examples show the effect of an increasing rate of occurrence on survival density, given the same number of events.

Example – number of events

```
DATA _NULL_;  
S5 = SDF("POISSON", 3, 1);  
PUT S5=;  
S6 = SDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=0.0189881569  
S6=0.0000832411
```

These two examples show the effect of an increasing number of events on the survival density, given the same rate of occurrence.

Example – decimal values

```
DATA _NULL_;
  S7 = SDF("POISSON", 1, 0.4);
  PUT S7=;
  S8 = SDF("POISSON", 1.1, 0);
  PUT S8=;
RUN;
```

This produces the following output:

```
S7=0.0615519356
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

LOGSDF – POISSON

Returns the natural logarithm of the survival density for a Poisson distribution, for a specified minimum number of events and mean rate of occurrence.

```
LOGSDF ( "POISSON" , n , lambda )
```

The natural logarithm of the survival density function (LOGSDF) for a Poisson distribution gives the natural logarithm of the probability of more than a specified number of events n occurring in a time period, given their mean rate of occurrence λ (*lambda*) in that same time period. The natural logarithm of the survival density function is defined as:

$$f(n; \lambda) = \log \left(1 - \sum_{i=0}^n \frac{e^{-\lambda} \lambda^i}{i!} \right)$$

Return type: Numeric

n

Type: Numeric

The start of the range of events for which to return the natural logarithm of the survival density. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $n \geq 0$ and must be integer

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as n , the number of events.

Restriction: $\lambda > 0$

If the argument is out of range, a missing value is returned.

Example – basic example

```
DATA _NULL_;  
S1 = LOGSDF("POISSON", 1, 1);  
PUT S1=;  
S2 = LOGSDF("POISSON", 1, 3);  
PUT S2=;  
RUN;
```

This produces the following output:

```
S1=-1.330893268  
S2=-0.22207946
```

The first example above gives the the natural logarithm of the survival density for 1 event, which is the natural logarithm of the probability of 2 or more events occurring. This is calculated with a rate of occurrence of 1. The second example triples the rate of occurrence of events, increasing the natural logarithm of the survival density.

Example – rate of occurrence

```
DATA _NULL_;  
S3 = LOGSDF("POISSON", 1, 3);  
PUT S3=;  
S4 = LOGSDF("POISSON", 1, 6);  
PUT S4=;  
RUN;
```

This produces the following output:

```
S3=-0.22207946  
S4=-0.017503563
```

These two examples show the effect of an increasing rate of occurrence on survival density, given the same number of events.

Example – number of events

```
DATA _NULL_;  
S5 = LOGSDF("POISSON", 3, 1);  
PUT S5=;  
S6 = LOGSDF("POISSON", 6, 1);  
PUT S6=;  
RUN;
```

This produces the following output:

```
S5=-3.963939816
S6=-9.39376875
```

These two examples show the effect of an increasing number of events on the survival density, given the same rate of occurrence.

Example – decimal values

```
DATA _NULL_;
  S7 = LOGSDF("POISSON", 1, 0.4);
  PUT S7=;
  S8 = LOGSDF("POISSON", 1.1, 0);
  PUT S8=;
RUN;
```

This produces the following output:

```
S7=-2.78787398
S8=.
```

These examples show the use of decimal values. The first example uses a decimal rate of occurrence, which is permitted. The second example uses a decimal number of occurrences, which is not permitted and therefore returns a missing value.

QUANTILE – POISSON

Returns the quantile for a Poisson distribution for a given cumulative density and mean rate of occurrence.

```
QUANTILE ( "POISSON" , q , lambda )
```

The quantile for a cumulative density q gives the smallest positive integer number of events n that results in that cumulative density in a time period, for a specified mean rate of occurrence λ (*lambda*) in that same time period.

$$\begin{cases} \text{if } q=0 & \text{return } lower \\ \text{if } q=1 & \text{return } upper \\ \text{otherwise} & \text{return } f(q) \end{cases}$$

$$f(q) = \inf \{x: q \leq \text{CDF}(x)\}$$

where $\inf\{x\}$ is the greatest lower bound of x (*infimum*) and $\text{CDF}(x)$ refers to the cumulative density function of the Poisson distribution (see section [CDF – POISSON](#) (page 1255)).

Return type: Numeric

q

Type: Numeric

The cumulative probability density for which to calculate the quantile. This should be specified for the same time period as *lambda*, the mean rate of occurrence.

Restriction: $0 \leq q < 1$

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

The mean rate of occurrence that defines this Poisson distribution. This should be specified for the same time period as *q*, the cumulative probability density.

Restriction: *lambda* > 0

If the argument is out of range, a missing value is returned.

Basic example

```
DATA _NULL_;
S1 = QUANTILE("POISSON", 0.6288369352, 4);
PUT S1=;
S2 = QUANTILE("POISSON", 0.9196986029, 1);
PUT S2=;
S3 = QUANTILE("POISSON", 0.8, 1);
PUT S3=;
RUN;
```

This produces the following output:

```
S1=5
S2=2
S3=2
```

The first two examples use the exact cumulative density required to give values of *n* of 5 and 2 respectively. The third example is the same as the second example, but the cumulative density is reduced from 0.92 to 0.8, and the value of *n* returned is the same, because it is still the smallest positive integer value corresponding to this cumulative density.

Example – cumulative density of zero

```
DATA _NULL_;
S4 = QUANTILE("POISSON", 0, 4);
PUT S4=;
RUN;
```

This produces the following output:

```
S4=0;
```

This example specifies a cumulative density of zero, which will always return a value for *n* of zero.

Example – invalid syntax

```
DATA _NULL_;
  S5 = QUANTILE("POISSON", 0, 0);
  PUT S5=;
  S6 = QUANTILE("POISSON", 1.1, 4);
  PUT S6=;
RUN;
```

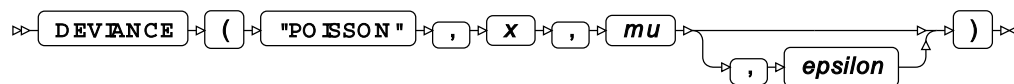
This produces the following output:

```
S5=.
S6=.
```

These examples are both invalid and return a missing value; the first example because of a zero value for λ , and the second because q is greater than 1.

DEVIANC – POISSON

Returns the deviance of the Poisson distribution at a specified point, based on the distribution mean.



Calculates the deviance, or goodness of fit, for the generalised linear model of the Poisson distribution at a nonnegative point x based on the distribution mean μ (mu). An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The distribution mean is then adjusted so that $\mu \geq \varepsilon$:

$$\text{if } \mu < \varepsilon \text{ set } \mu = \varepsilon$$

This adjusted value of μ is used in the subsequent calculation of the deviance.

$$\begin{cases} \text{if } x = 0 & \text{return } 2\mu \\ \text{otherwise} & \text{return } 2\left(x \log \frac{x}{\mu} - (x - \mu)\right) \end{cases}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

mu**Type:** Numeric

The distribution mean.

Expected: $\mu > 0$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$ **Expected:** $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Examples – applying correction to the distribution mean

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("POISSON", 0.1, 0.0007, 0.0005);  
  PUT g1=;  
  g2 = DEVIANCE("POISSON", 0.1, 0.0007, 0.0010);  
  PUT g2=;  
  g3 = DEVIANCE("POISSON", 0.1, 0.0007, 0.0015);  
  PUT g3=;  
  g4 = DEVIANCE("POISSON", 0.1, 0.0007          );  
  PUT g4=;  
RUN;
```

This produces the following output:

```
g1=0.793769026  
g2=0.7230340372  
g3=0.6429410156  
g4=0.793769026
```

The value of the distribution mean is not corrected in the first example because $\mu > \varepsilon$. However, this condition does not hold in the second and third example, and correction is applied: $\mu = \varepsilon$. This corrected value is used for calculation, yielding different results.

In the fourth example the ε parameter is omitted, so the default value of $\varepsilon = 10^{-12}$ is used. Here, as in the first example, $\mu > \varepsilon$, so no correction is required.

Examples – parallel scaling

In these examples, the deviance is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = DEVIANCE("POISSON", 0.01, 0.07);  
  PUT g1=;  
  g2 = DEVIANCE("POISSON", 0.1 , 0.7 );  
  PUT g2=;  
  g3 = DEVIANCE("POISSON", 1 , 7 );  
  PUT g3=;  
RUN;
```

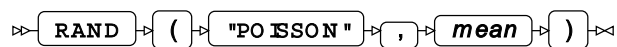
This produces the following output:

```
g1=0.081081797  
g2=0.8108179702  
g3=8.1081797019
```

For the Poisson distribution, when both the point of measurement x and the distribution mean μ are scaled with the same factor, the deviance is scaled with the same factor too.

RAND – POISSON

Returns a random number from the Poisson distribution based on the mean. This function is similar to RANPOI and CALL RANPOI.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is a positive integer or zero.

mean

Type: Numeric

The mean of the distribution.

Example

In this example, a random number from the Poisson distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("POISSON", 21);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
22  
25  
18  
22  
23
```

Running the DATA step again produces the following output.

```
The random numbers are:  
18  
24  
22  
18  
19
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  CALL STREAMINIT(9);  
  DO i = 1 TO 5;  
    result = RAND("POISSON", 21);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
23  
18  
26  
23  
13
```

Running the DATA step again produces the same output.

RANPOI

Returns a random number from the Poisson distribution based on the mean. This function is similar to RAND – POISSON and CALL RANPOI.



The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is a positive integer or zero.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

mean

Type: Numeric

The mean of the distribution.

This must be a positive number. If a negative number is used, a missing value is returned from the function, and a note indicating that the argument is invalid is written to the log.

Example

In this example, a random number from the Poisson distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RANPOI(10, 50);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
57
54
72
52
44
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RANPOI(0, 50);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
40
48
44
62
44
```

Running the DATA step again produces the following output.

```
The random numbers are:
49
54
56
41
58
```

CALL RANPOI

Returns a random number from the Poisson distribution based on the mean. This routine is similar to function RAND – POISSON and RANPOI.

```
CALL RANPOI ( seed , mean , x ) ;
```

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a DATA step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same `DATA` step. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is a positive integer or zero.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

mean

Type: Numeric

The mean of the distribution.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Poisson distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  DO i = 1 TO 5;  
    call ranpoi(10, 1, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
2  
1  
5  
1  
0
```

Running the `DATA` step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    call ranpoi(0, 1, ranN);
    PUT ranN;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0
1
1
1
0
```

Running the DATA step again produces the following output.

```
The random numbers are:
0
3
2
1
0
```

Power distribution

Functions for the Power distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

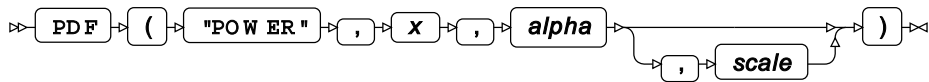
- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – POWER ↗	1275
Returns the probability density of the Power distribution. This function is an alias of PMF – POWER.	
PMF – POWER ↗	1276
Returns the probability mass of the Power distribution. This function is an alias of PDF – POWER.	

LOGPDF – POWER ↗	1276
Returns the natural logarithm of the probability density of the Power distribution. This function is an alias of LOGPMF – POWER.	
LOGPMF – POWER ↗	1277
Returns the natural logarithm of the probability mass of the Power distribution. This function is an alias of LOGPDF – POWER.	
CDF – POWER ↗	1277
Returns the cumulative density of the Power distribution.	
LOGCDF – POWER ↗	1278
Returns the natural logarithm of the cumulative density of the Power distribution.	
SDF – POWER ↗	1278
Returns the survival of the Power distribution.	
LOGSDF – POWER ↗	1279
Returns the natural logarithm of the survival of the Power distribution.	
QUANTILE – POWER ↗	1280
Returns the quantile of the Power distribution.	
RAND – POWER ↗	1280
Returns a random number from the Power distribution based on the shape.	

PDF – POWER

Returns the probability density of the Power distribution. This function is an alias of PMF – POWER.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

alpha

Type: Numeric

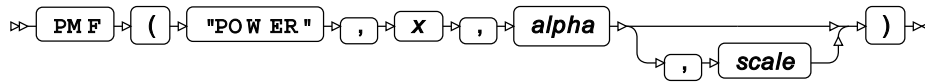
scale

Optional argument

Type: Numeric

PMF – POWER

Returns the probability mass of the Power distribution. This function is an alias of PDF – POWER.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

alpha

Type: Numeric

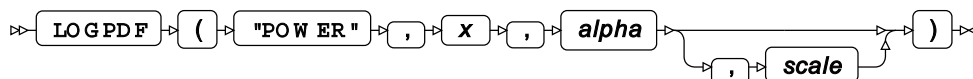
scale

Optional argument

Type: Numeric

LOGPDF – POWER

Returns the natural logarithm of the probability density of the Power distribution. This function is an alias of LOGPMF – POWER.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

alpha

Type: Numeric

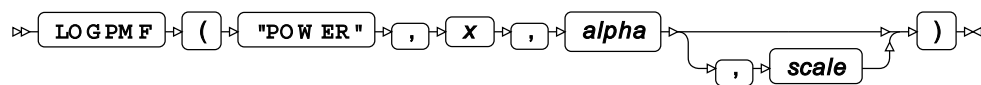
scale

Optional argument

Type: Numeric

LOGPMF – POWER

Returns the natural logarithm of the probability mass of the Power distribution. This function is an alias of LOGPDF – POWER.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

alpha

Type: Numeric

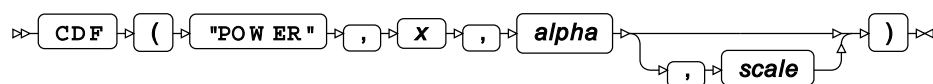
scale

Optional argument

Type: Numeric

CDF – POWER

Returns the cumulative density of the Power distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

alpha

Type: Numeric

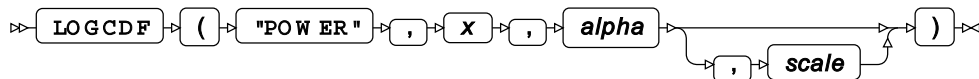
scale

Optional argument

Type: Numeric

LOGCDF – POWER

Returns the natural logarithm of the cumulative density of the Power distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

alpha

Type: Numeric

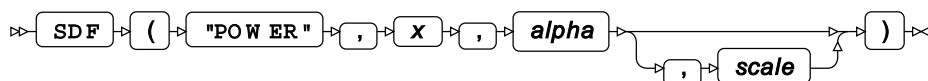
scale

Optional argument

Type: Numeric

SDF – POWER

Returns the survival of the Power distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

alpha

Type: Numeric

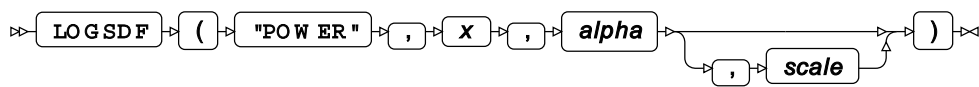
scale

Optional argument

Type: Numeric

LOGSDF – POWER

Returns the natural logarithm of the survival of the Power distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

alpha

Type: Numeric

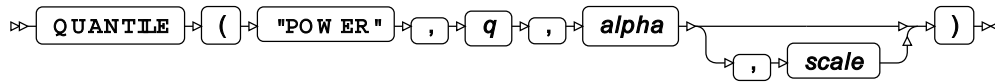
scale

Optional argument

Type: Numeric

QUANTILE – POWER

Returns the quantile of the Power distribution.



Return type: Numeric

q

Type: Numeric

alpha

Type: Numeric

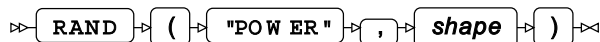
scale

Optional argument

Type: Numeric

RAND – POWER

Returns a random number from the Power distribution based on the shape.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\[link\]](#) (page 777) before this function.

Return type: Numeric

The return value is positive.

shape

Type: Numeric

The shape of the distribution.

Example

In this example, a random number from the Power distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("POWER", 5);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.9213691046  
0.8019731289  
0.784980668  
0.7568561993  
0.9939329395
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.9315627133  
0.7662125504  
0.9704090536  
0.8353239284  
0.8949201734
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  CALL STREAMINIT(12);  
  DO i = 1 TO 5;  
    result = RAND("POWER", 5);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.8977992771  
0.9262939355  
0.801520703  
0.9658946335  
0.8576218172
```

Running the DATA step again produces the same output.

Rayleigh distribution

Functions for the Rayleigh distribution.

RAND – RAYLEIGH [↗](#) 1282
Returns a random number from the Rayleigh distribution.

RAND – RAYLEIGH

Returns a random number from the Rayleigh distribution.

➤ **RAND** ➤ (➤ **"RAYLEIGH"** ➤) ➤ ➤

The distribution is parameterised using a scale of 2.0.

This function does not take any variable arguments.

Each time you execute this function within a **DATA** step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same **DATA** step produce different sequences of random numbers. To initialise the random stream, use **CALL STREAMINIT** [↗](#) (page 777) before this function.

Return type: Numeric

The return value is positive.

Example

In this example, a random number from the Rayleigh distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("RAYLEIGH");  
    PUT result;  
  END;  
RUN;
```


This produces the following output:

```
The random numbers are:
1.8860668102
1.6524286007
1.8905569244
0.983327276
1.5175924268
```

Running the DATA step again produces the following output.

```
The random numbers are:
2.0536582778
1.9443191378
0.9008592528
1.3204158219
0.362170589
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  PUT "The random numbers are:";
  CALL STREAMINIT(12);
  DO i = 1 TO 5;
    result = RAND("RAYLEIGH");
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1.0383099623
0.8750066843
1.4874289125
0.5890715218
1.239322594
```

Running the DATA step again produces the same output.

Student's T distribution

Functions for the Student's T distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

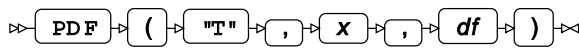
Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – T ↗	1284
Returns the probability density of the Student's T distribution. This function is an alias of PMF – T.	
PMF – T ↗	1285
Returns the probability mass of the Student's T distribution. This function is an alias of PDF – T.	
LOGPDF – T ↗	1285
Returns the natural logarithm of the probability density of the Student's T distribution. This function is an alias of LOGPMF – T.	
LOGPMF – T ↗	1286
Returns the natural logarithm of the probability mass of the Student's T distribution. This function is an alias of LOGPDF – T.	
CDF – T ↗	1286
Returns the cumulative density of the Student's T distribution.	
LOGCDF – T ↗	1286
Returns the natural logarithm of the cumulative density of the Student's T distribution.	
SDF – T ↗	1287
Returns the survival of the Student's T distribution.	
LOGSDF – T ↗	1287
Returns the natural logarithm of the survival of the Student's T distribution.	
QUANTILE – T ↗	1288
Returns the quantile of the Student's T distribution.	
PROBT ↗	1288
TINV ↗	1289
RAND – T ↗	1289
Returns a random number from the Student's T distribution based on the number of degrees of freedom.	

PDF – T

Returns the probability density of the Student's T distribution. This function is an alias of PMF – T.



Return type: Numeric

x

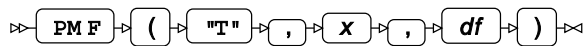
Type: Numeric

The point at which to calculate the probability density.

df**Type:** Numeric

PMF – T

Returns the probability mass of the Student's T distribution. This function is an alias of PDF – T.

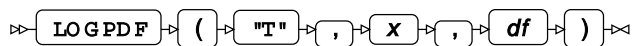
**Return type:** Numeric**x****Type:** Numeric

The point at which to calculate the probability mass.

df**Type:** Numeric

LOGPDF – T

Returns the natural logarithm of the probability density of the Student's T distribution. This function is an alias of LOGPMF – T.

**Return type:** Numeric**x****Type:** Numeric

The point at which to calculate the natural logarithm of the probability density.

df**Type:** Numeric

LOGPMF – T

Returns the natural logarithm of the probability mass of the Student's T distribution. This function is an alias of LOGPDF – T.

⇒ LOGPMF ⇒ (⇒ "T" ⇒ , ⇒ x ⇒ , ⇒ df ⇒) ⇒

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

df

Type: Numeric

CDF – T

Returns the cumulative density of the Student's T distribution.

⇒ CDF ⇒ (⇒ "T" ⇒ , ⇒ x ⇒ , ⇒ df ⇒) ⇒

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

df

Type: Numeric

LOGCDF – T

Returns the natural logarithm of the cumulative density of the Student's T distribution.

⇒ LOGCDF ⇒ (⇒ "T" ⇒ , ⇒ x ⇒ , ⇒ df ⇒) ⇒

Return type: Numeric

x

Type: Numeric

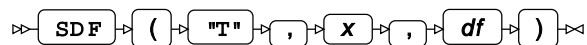
The point at which to calculate the natural logarithm of the cumulative density.

df

Type: Numeric

SDF – T

Returns the survival of the Student's T distribution.



Return type: Numeric

x

Type: Numeric

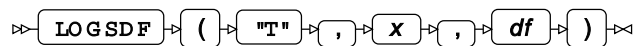
The point at which to calculate the survival.

df

Type: Numeric

LOGSDF – T

Returns the natural logarithm of the survival of the Student's T distribution.



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

df

Type: Numeric

QUANTILE – T

Returns the quantile of the Student's T distribution.



Return type: Numeric

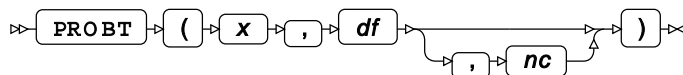
q

Type: Numeric

df

Type: Numeric

PROBT



Return type: Numeric

x

Type: Numeric

df

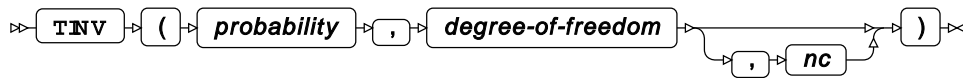
Type: Numeric

nc

Optional argument

Type: Numeric

TINV



Return type: Numeric

probability

Type: Numeric

degree-of-freedom

Type: Numeric

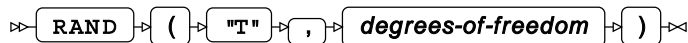
nc

Optional argument

Type: Numeric

RAND – T

Returns a random number from the Student's T distribution based on the number of degrees of freedom.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\[link\]](#) (page 777) before this function.

Return type: Numeric

degrees-of-freedom

Type: Numeric

The number of degrees of freedom.

Example

In this example, a random number from the Student's T distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("T", 10);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.1970340178  
2.2774759968  
0.7133708614  
1.6070218831  
-0.539787121
```

Running the DATA step again produces the following output.

```
The random numbers are:  
1.4278503855  
-1.424301771  
-0.525567429  
1.1571239401  
-0.379811436
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(20);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("T", 10);  
    PUT result;  
  END;  
run
```

This produces the following output:

```
The random numbers are:  
-0.037470678  
0.3981560865  
1.3624854501  
2.4989899483  
-0.798725141
```

Running the DATA step again produces the same output.

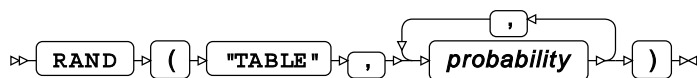
Table distribution

Functions and `CALL` routines for the Table distribution.

<code>RAND</code> – <code>TABLE</code> ↗	1291
Returns a random number from the Table distribution based on the probabilities for each category. This function is similar to <code>RANTBL</code> and <code>CALL RANTBL</code> .	
<code>RANTBL</code> ↗	1293
Returns a random number from the Table distribution based on the probabilities for each category. This function is similar to <code>RAND – TABLE</code> and <code>CALL RANTBL</code> .	
<code>CALL RANTBL</code> ↗	1298
Returns a random number from the Table distribution based on the probabilities for each category. This routine is similar to function <code>RAND – TABLE</code> and <code>RANTBL</code> .	

RAND – TABLE

Returns a random number from the Table distribution based on the probabilities for each category. This function is similar to `RANTBL` and `CALL RANTBL`.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

The number returned will be the number randomly of a selected category. If the probabilities sum to 1, an ordinal number will be applied to each probability in the order they are specified. For example, in `rantbl(5, 0.75, 0.25)` the 0.75 probability will be assigned to category 1, and 0.25 assigned to category 2. The random numbers returned would then be either 1 or 2, and would be expected to occur, on average, 75% and 25% of the time respectively. If you specified `rantbl(5, 0.25, 0.25, 0.5, 0.25)`:

- The first 0.25 probability will be assigned to category 1
- The second 0.25 probability assigned to category 2
- The 0.5 probability assigned to category 3
- The final 0.25 probability to category 4

The random numbers returned would then be in the range 1 through 4; the numbers 1, 2 and 4 would appear, on average, 25% of the time, while the number 3 would appear, on average 50% of the time.

If the numbers sum to less than 1, then a category is created that represents the probability required to sum to 1. For example, `rantbl(5,0.5,0.2,0.1)` would result in four categories, with the number 4 assigned to the probability 0.2 that would be required to sum the probabilities to 1.

If the numbers sum to greater than 1, the result depends on which of the arguments caused the probability to sum to 1. If it is the final argument, then the category assigned will only represent the probability required to sum to 1. For example, `rantbl(5,0.5,0.3,0.3)` would result in three categories, with the number 3 assigned the probability 0.2 that would be required to sum the probabilities to 1; that is, 3 would only be expected to be returned 20% of the time, rather than 30% of the time. If the probabilities sum to 1 before the final argument, then the arguments beyond the argument that causes the probabilities to sum to 1 are discarded. For example, if you specified `rantbl(5,0.6,0.6,0.3,0.2)`, only two categories would be specified: category 1 with a probability of occurring, on average, 60% of the time, and 2 occurring, on average, 40% of the time.

Return type: Numeric

probability

Type: Numeric

A probability from the table probability.

Example

In this example, a random number from the Table distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("TABLE",0.2,0.5,0.2,0.1);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
1
3
3
3
1
```

Running the `DATA` step again produces the following output.

```
The random numbers are:
3
2
2
3
2
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  PUT "The random numbers are:";
  CALL STREAMINIT(9);
  DO i = 1 TO 5;
    result = result = RAND("TABLE",0.7);
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
3
2
3
2
1
2
2
2
4
2
2
```

Running the `DATA` step again produces the same output.

RANTBL

Returns a random number from the Table distribution based on the probabilities for each category. This function is similar to `RAND – TABLE` and `CALL RANTBL`.



The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

The number returned will be the number of a randomly selected category. If the probabilities sum to 1, an ordinal number will be applied to each probability in the order they are specified. For example, in `rantbl(5, 0.75, 0.25)` the 0.75 probability will be assigned to category 1, and 0.25 assigned to category 2. The random numbers returned would then be either 1 or 2, and would be expected to occur, on average, 75% and 25% of the time respectively. If you specified `rantbl(5, 0.25, 0.25, 0.5, 0.25)`:

- The first 0.25 probability will be assigned to category 1
- The second 0.25 probability assigned to category 2
- The 0.5 probability assigned to category 3
- The final 0.25 probability to category 4

The random numbers returned would then be in the range 1 through 4; the numbers 1, 2 and 4 would appear, on average, 25% of the time, while the number 3 would appear, on average 50% of the time.

If the numbers sum to less than 1, then a category is created that represents the probability required to sum to 1. For example, `rantbl(5, 0.5, 0.2, 0.1)` would result in four categories, with the number 4 assigned to the probability 0.2 that would be required to sum the probabilities to 1.

If the numbers sum to greater than 1, the result depends on which of the arguments caused the probability to sum to 1. If it is the final argument, then the category assigned will only represent the probability required to sum to 1. For example, `rantbl(5, 0.5, 0.3, 0.3)` would result in three categories, with the number 3 assigned the probability 0.2 that would be required to sum the probabilities to 1; that is, 3 would only be expected to be returned 20% of the time, rather than 30% of the time. If the probabilities sum to 1 before the final argument, then the arguments beyond the argument that causes the probabilities to sum to 1 are discarded. For example, if you specified `rantbl(5, 0.6, 0.6, 0.3, 0.2)`, only two categories would be specified: category 1 with a probability of occurring, on average, 60% of the time, and 2 occurring, on average, 40% of the time.

Return type: Numeric

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

prob

Type: Numeric

A discrete probability.

Basic example

In this example, a random number from the Table distribution is returned on each iteration of the loop. The result is written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 10;  
    result = RANTBL(5, 0.25, 0.25, 0.5, 0.25);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
3  
3  
3  
2  
3  
3  
3  
3  
3  
1  
2
```

Running the DATA step again produces the following output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANTBL(0, 0.7, 0.2, 0.1);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
1  
1  
2  
1  
3
```

Running the DATA step again produces the following output.

```
The random numbers are:  
3  
1  
1  
1  
1
```

Example – probabilities sum to greater than 1

In this example, probabilities sum to more than 1. The result is written to the log.

```
DATA _NULL_;
  o = 0;
  x = 0;
  y = 0;

  average_x=0;
  average_y=0;

  toax=0;
  toay=0;
  overall_average_x=0;
  overall_average_y=0;
  m = 0;

  DO k = 1 TO 100;

    DO i = 1 TO 500;
      result = RANTBL(0, 0.6, 0.6, 0.3, 0.2);
      if result = 1 then x = x + 1;
      if result = 2 then y = y + 1;
      o = o + 1;
    END;

    average_x = x/o*100;
    average_y = y/o*100;

    overall_average_x = overall_average_x + average_x;
    overall_average_y = overall_average_y + average_y;

    m = m + 1;

  END;

  toax = overall_average_x/m;
  toay = overall_average_y/m;

  put "Average occurrence of 1 = " toax;
  put "Average occurrence of 2 = " toay;

RUN;
```

This produces the following output:

```
Average occurrence of 1 = 60.109367963
Average occurrence of 2 = 39.890632037
```

The last two probabilities in the list of arguments provided to the function are ignored, as the list of probabilities sum to greater than 1. The second argument (0.6) is instead given the value 0.4 so that the probabilities sum to 1. Only two values are then returned, 1 and 2. 1 will be returned, on average, 60% of the time, matching the specified argument of 0.6, while the 2 will be returned 40% of the time, matching the redefined second argument. The values returned by the example match these averages.

Example – probabilities sum to less than 1

In this example, probabilities sum to less than 1. The result is written to the log.

```
DATA _NULL_;

m = 0;
o = 0;
v = 0;
w = 0;
x = 0;
y = 0;
average_x = 0;
average_y = 0;
average_v = 0;
average_w = 0;
toav = 0;
toaw = 0;
toax = 0;
toay = 0;
overall_average_x = 0;
overall_average_y = 0;
overall_average_v = 0;
overall_average_w = 0;

DO k = 1 TO 100;

  DO i = 1 TO 500;
    result = RANTBL(0, 0.4, 0.2, 0.2);
    if result = 1 then x = x + 1;
    if result = 2 then y = y + 1;
    if result = 3 then v = v + 1;
    if result = 4 then w = w + 1;
    o = o + 1;
  END;
  average_x = x/o*100;
  average_y = y/o*100;
  average_v = v/o*100;
  average_w = w/o*100;

  overall_average_x = overall_average_x + average_x;
  overall_average_y = overall_average_y + average_y;
  overall_average_w = overall_average_w + average_w;
  overall_average_v = overall_average_v + average_v;
  m = m + 1;

END;

toax = overall_average_x/m;
toay = overall_average_y/m;
toav = overall_average_v/m;
toaw = overall_average_w/m;

put "Average occurrence of 1 = " toax;
put "Average occurrence of 2 = " toay;
put "Average occurrence of 3 = " toav;
put "Average occurrence of 4 = " toaw;

RUN;
```

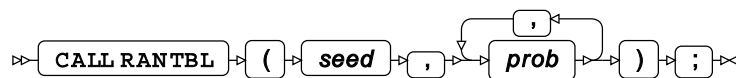
This produces the following output:

```
Average occurrence of 1 = 39.654043034
Average occurrence of 2 = 20.068023606
Average occurrence of 3 = 20.077069785
Average occurrence of 4 = 20.200863575
```

As the results show, although only three arguments are supplied, results for four categories are returned. Because the probabilities supplied as arguments to the function sum to less than 1, a fourth argument is assumed by the function, having a value that causes all probabilities to sum to 1. In this example, four values are then returned, 1 through 4, where 4 which will be returned, on average, 20% of the time; that is, the assumed argument is given a probability of 0.2, to make the arguments sum to 1. The values returned by the example the average number of times each category is returned.

CALL RANTBL

Returns a random number from the Table distribution based on the probabilities for each category. This routine is similar to function RAND – TABLE and RANTBL.



Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a **DATA** step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the **DATA** step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same **DATA** step. To generate a different sequence of random numbers each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The last argument in the list of probabilities is the output parameter where the result is stored.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

prob

Type: Numeric

A discrete probability.

The last argument in the list is where the result is stored. The other arguments are not updated.

Basic examples

In this example, a random number from the Table distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 10;  
    result = CALL RANTBL(5, 0.25, 0.25, 0.5, 0.25);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
4  
3  
4  
1  
3  
3  
3  
3  
3  
1  
1
```

Running the DATA step again produces the following output.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 10;  
    call rantbl(0, 0.4, 0.2, 0.3, 0.1, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
4  
3  
3  
1  
1  
2  
1  
1  
3  
3
```

Running the DATA step again produces the following output.

```
TThe random numbers are:  
2  
1  
3  
1  
1  
1  
4  
2  
1  
3
```

Example – probabilities sum to greater than 1

In this example, the probabilities sum to more than 1. The result is written to the log.

```
DATA _NULL_;  
  o = 0;  
  x = 0;  
  y = 0;  
  toax=0;  
  toay=0;  
  overall_average_x=0;  
  overall_average_y=0;  
  m = 0;  
  
  DO k = 1 TO 100;  
  
    DO i = 1 TO 500;  
      result = CALL RANTBL(0, 0.6, 0.6, 0.3, 0.2);  
      if result = 1 then x = x + 1;  
      if result = 2 then y = y + 1;  
      o = o + 1;  
    END;  
  
    overall_average_x = overall_average_x + x/o*100;  
    overall_average_y = overall_average_y + y/o*100;  
  
    m = m + 1;  
  
  END;  
  
  toax = overall_average_x/m;  
  toay = overall_average_y/m;  
  
  put "Average occurrence of 1 = " toax;  
  put "Average occurrence of 2 = " toay;  
  
RUN;
```

This produces the following output:

```
Average occurrence of 1 = 60.109367963  
Average occurrence of 2 = 39.890632037
```

The last two probabilities in the list of arguments provided to the function are ignored, as the list of probabilities sum to greater than 1. The second argument (0.6) is instead given the value 0.4 so that the probabilities sum to 1. Only two values are then returned, 1 and 2. 1 will be returned, on average, 60% of the time, matching the specified argument of 0.6, while the 2 will be returned 40% of the time, matching the redefined second argument. The values returned by the example match these averages.

Example – probabilities sum to less than 1

In this example, probabilities sum to less than 1. The result is written to the log.

```
DATA _NULL_;

m = 0;
o = 0;
v = 0;
w = 0;
x = 0;
y = 0;
toav = 0;
toaw = 0;
toax = 0;
toay = 0;
overall_average_x = 0;
overall_average_y = 0;
overall_average_v = 0;
overall_average_w = 0;

DO k = 1 TO 100;

  DO i = 1 TO 500;
    call randtbl(0, 0.4, 0.2, 0.2, nN);
    if nN = 1 then x = x + 1;
    if nN = 2 then y = y + 1;
    if nN = 3 then v = v + 1;
    if nN = 4 then w = w + 1;
    o = o + 1;
  END;

  overall_average_x = overall_average_x + x/o*100;
  overall_average_y = overall_average_y + y/o*100;
  overall_average_w = overall_average_w + w/o*100;
  overall_average_v = overall_average_v + v/o*100;

  m = m + 1;

END;

toax = overall_average_x/m;
toay = overall_average_y/m;
toav = overall_average_v/m;
toaw = overall_average_w/m;

put "Average occurrence of 1 = " toax;
put "Average occurrence of 2 = " toay;
put "Average occurrence of 3 = " toav;
put "Average occurrence of 4 = " toaw;

RUN;
```

This produces the following output:

```
Average occurrence of 1 = 40.554983457
Average occurrence of 2 = 20.050067559
Average occurrence of 3 = 19.616092583
Average occurrence of 4 = 19.778856401
```

As the results show, although only three arguments are supplied, results are returned for four categories. Because the probabilities supplied as arguments to the function sum to less than 1, a fourth argument is assumed by the function, having a value that causes all probabilities to sum to 1. In this example, four values are then returned, 1 through 4, where 4 which will be returned, on average, 20% of the time; that is, the assumed argument is given a probability of 0.2, to make the arguments sum to 1. The values returned by the example the average number of times each category is returned.

Triangular distribution

Functions and CALL routines for the Triangular distribution.

RAND – TRIANGLE ↗	1302
Returns a random number from the Triangular distribution based on the mode. This function is similar to RANTRI and CALL RANTRI.	
RANTRI ↗	1304
Returns a random number from the Triangular distribution based on the mode. This function is similar to RAND – TRIANGLE and CALL RANTRI.	
CALL RANTRI ↗	1306
Returns a random number from the Triangular distribution based on the mode. This routine is similar to function RAND – TRIANGLE and RANTRI.	

RAND – TRIANGLE

Returns a random number from the Triangular distribution based on the mode. This function is similar to RANTRI and CALL RANTRI.

```
➡ RAND ( "TRIANGLE" , mode ) ➡
```

Each time you execute this function within a DATA step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same DATA step produce different sequences of random numbers. To initialise the random stream, use CALL STREAMINIT [↗](#) (page 777) before this function.

Return type: Numeric

The return value is between 0 and 1, inclusive.

mode

Type: Numeric

The mode of the distribution.

Example

In this example, a random number from the Triangular distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("TRIANGLE",0.78);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.2909909069  
0.5297777426  
0.2778985607  
0.7423049924  
0.7418666133
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.4241274196  
0.5446775117  
0.7673564187  
0.4422557575  
0.7960401493
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;  
  CALL STREAMINIT(9);  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RAND("TRIANGLE",0.78);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.7453581494  
0.4475660965  
0.8270281223  
0.7353654381  
0.1505576581
```

Running the `DATA` step again produces the same output.

RANTRI

Returns a random number from the Triangular distribution based on the mode. This function is similar to `RAND – TRIANGLE` and `CALL RANTRI`.

```
RANTRI ( seed , mode )
```

The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is between 0 and 1, inclusive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

mode

Type: Numeric

The mode for the triangular distribution.

Example

In this example, a random number from the Triangular distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANTRI(50, 0.34);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.2904542769  
0.4280975752  
0.9238560636  
0.8591287731  
0.1650570317
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = RANTRI(0, 0.34);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.9258540423  
0.3679089951  
0.8142370697  
0.4901708232  
0.5053367342
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.2809093445  
0.2274784681  
0.3597153279  
0.3133674246  
0.7264610997
```

CALL RANTRI

Returns a random number from the Triangular distribution based on the mode. This routine is similar to function RAND – TRIANGLE and RANTRI.

```
CALL RANTRI ( seed , mode , x ) ;
```

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a DATA step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the DATA step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same DATA step. To generate a different sequence of random numbers each time the DATA step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is between 0 and 1, inclusive.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

mode

Type: Numeric

The mode of distribution.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Triangular distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    call rantri(50, 0.34, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.2904542769  
0.4280975752  
0.9238560636  
0.8591287731  
0.1650570317
```

Running the DATA step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result = CALL RANTRI(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.461364197  
0.8069769932  
0.4751027529  
0.5706897081  
0.412730761
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.5723194432  
0.357089012  
0.5004735172  
0.1554828266  
0.3989297724
```

Tweedie distribution

Functions for the Tweedie distribution.

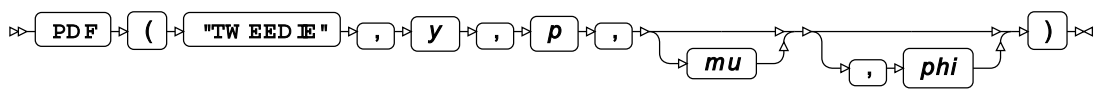
Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – TWEEDIE ↗	1308
PMF – TWEEDIE ↗	1309
LOGPDF – TWEEDIE ↗	1309
LOGPMF – TWEEDIE ↗	1310
CDF – TWEEDIE ↗	1311
LOGCDF – TWEEDIE ↗	1311
SDF – TWEEDIE ↗	1312
LOGSDF – TWEEDIE ↗	1312
QUANTILE – TWEEDIE ↗	1313

PDF – TWEEDIE



Return type: Numeric

y

Type: Numeric

p

Type: Numeric

mu

Optional argument

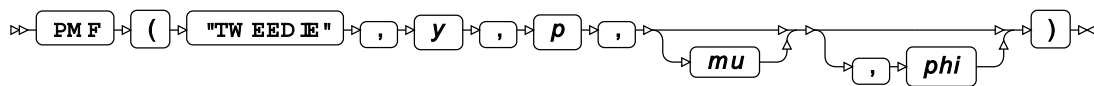
Type: Numeric

phi

Optional argument

Type: Numeric

PMF – TWEEDIE

**Return type:** Numeric***y*****Type:** Numeric***p*****Type:** Numeric***mu***

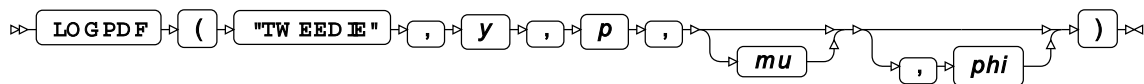
Optional argument

Type: Numeric***phi***

Optional argument

Type: Numeric

LOGPDF – TWEEDIE

**Return type:** Numeric

y

Type: Numeric

p

Type: Numeric

mu

Optional argument

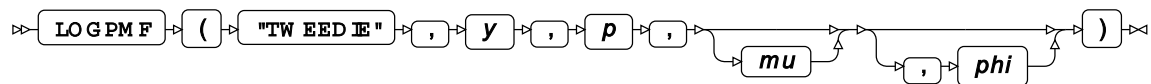
Type: Numeric

phi

Optional argument

Type: Numeric

LOGPMF – TWEEDIE



Return type: Numeric

y

Type: Numeric

p

Type: Numeric

mu

Optional argument

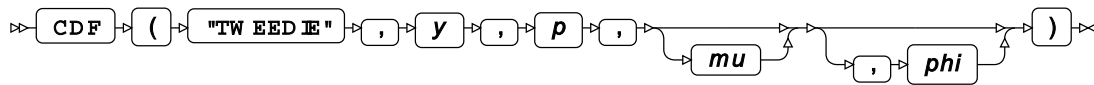
Type: Numeric

phi

Optional argument

Type: Numeric

CDF – TWEEDIE



Return type: Numeric

y

Type: Numeric

p

Type: Numeric

mu

Optional argument

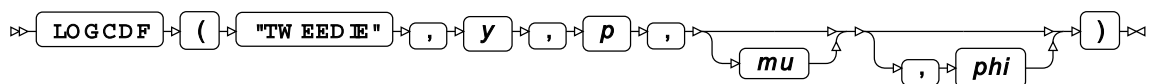
Type: Numeric

phi

Optional argument

Type: Numeric

LOGCDF – TWEEDIE



Return type: Numeric

y

Type: Numeric

p

Type: Numeric

mu

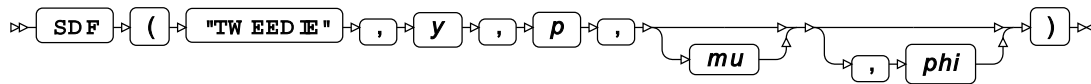
Optional argument

Type: Numeric***phi***

Optional argument

Type: Numeric

SDF – TWEEDIE

**Return type:** Numeric***y*****Type:** Numeric***p*****Type:** Numeric***mu***

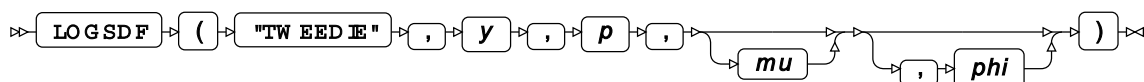
Optional argument

Type: Numeric***phi***

Optional argument

Type: Numeric

LOGSDF – TWEEDIE



Return type: Numeric

y

Type: Numeric

p

Type: Numeric

mu

Optional argument

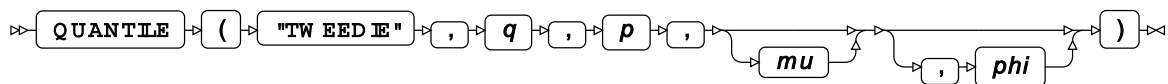
Type: Numeric

phi

Optional argument

Type: Numeric

QUANTILE – TWEEDIE



Return type: Numeric

q

Type: Numeric

p

Type: Numeric

mu

Optional argument

Type: Numeric

phi

Optional argument

Type: Numeric

Uniform distribution

Functions and `CALL` routines for the Uniform distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – UNIFORM [↗](#)..... 1315

$$\frac{1}{upper-lower}$$

Returns the probability density of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of PMF – UNIFORM.

PMF – UNIFORM [↗](#)..... 1317

$$\frac{1}{upper-lower}$$

Returns the probability mass of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of PDF – UNIFORM.

LOGPDF – UNIFORM [↗](#)..... 1319

$$\log \frac{1}{upper-lower}$$

Returns the natural logarithm of the probability density of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of LOGPMF – UNIFORM.

LOGPMF – UNIFORM [↗](#)..... 1320

$$\log \frac{1}{upper-lower}$$

Returns the natural logarithm of the probability mass of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of LOGPDF – UNIFORM.

CDF – UNIFORM [↗](#)..... 1322

$$\frac{x-lower}{upper-lower}$$

Returns the cumulative density of the Uniform distribution at a specified point within the specified domain limits.

LOGCDF – UNIFORM [↗](#)..... 1323

$$\log \frac{x-lower}{upper-lower}$$

Returns the natural logarithm of the cumulative density of the Uniform distribution at a specified point within the specified domain limits.

SDF – UNIFORM [↗](#)..... 1325

$$\frac{\text{upper} - x}{\text{upper} - \text{lower}}$$

Returns the survival of the Uniform distribution at a specified point within the specified domain limits.

LOGSDF – UNIFORM [↗](#)..... 1326

$$\log \frac{\text{upper} - x}{\text{upper} - \text{lower}}$$

Returns the natural logarithm of the survival of the Uniform distribution at a specified point within the specified domain limits.

QUANTILE – UNIFORM [↗](#)..... 1328

$$f(q) = \text{lower} + q(\text{upper} - \text{lower})$$

Returns the quantile of the Uniform distribution for a specified probability value within the specified domain limits.

RAND – UNIFORM [↗](#)..... 1329
Returns a random number from the Uniform distribution. This function is similar to RANUNI, UNIFORM and CALL RANUNI.

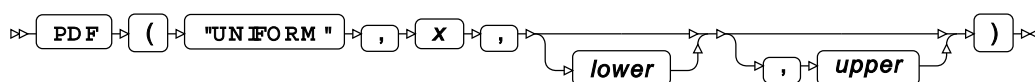
RANUNI [↗](#)..... 1331
Returns a random number from the Uniform distribution. This function is an alias of UNIFORM. This function is similar to RAND – UNIFORM and CALL RANUNI.

UNIFORM [↗](#)..... 1333
Returns a random number from the Uniform distribution. This function is an alias of RANUNI. This function is similar to RAND – UNIFORM and CALL RANUNI.

CALL RANUNI [↗](#)..... 1333
Returns a random number from the Uniform distribution. This routine is similar to function RAND – UNIFORM, UNIFORM and RANUNI.

PDF – UNIFORM

Returns the probability density of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of PMF – UNIFORM.



Calculates the probability density function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x < lower \text{ or } x > upper & \text{return } 0 \\ \text{otherwise} & \text{return } f(x) \end{cases}$$

$$f(x) = \frac{1}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

$lower$

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

$upper$

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability density of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = PDF ("UNIFORM", -0.5, -1, 4);
  PUT s1=;
  s2 = PDF ("UNIFORM", 3, -1, 4);
  PUT s2=;
  s3 = PDF ("UNIFORM", -3, -1, 4);
  PUT s3=;
  s4 = PDF ("UNIFORM", 7, -1, 4);
  PUT s4=;
RUN;
```

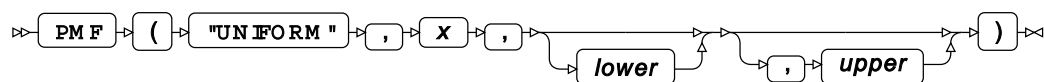
This produces the following output:

```
s1=0.2
s2=0.2
s3=0
s4=0
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

PMF – UNIFORM

Returns the probability mass of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of PDF – UNIFORM.



Calculates the probability mass function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x < lower \text{ or } x > upper & \text{return } 0 \\ \text{otherwise} & \text{return } f(x) \end{cases}$$

$$f(x) = \frac{1}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the probability mass of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = PMF ("UNIFORM", -0.5, -1, 4);  
  PUT s1=;  
  s2 = PMF ("UNIFORM", 3, -1, 4);  
  PUT s2=;  
  s3 = PMF ("UNIFORM", -3, -1, 4);  
  PUT s3=;  
  s4 = PMF ("UNIFORM", 7, -1, 4);  
  PUT s4=;  
RUN;
```

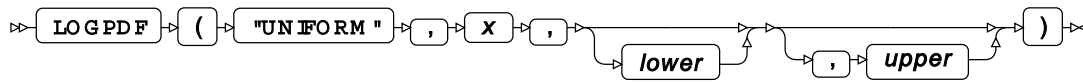
This produces the following output:

```
s1=0.2  
s2=0.2  
s3=0  
s4=0
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

LOGPDF – UNIFORM

Returns the natural logarithm of the probability density of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of LOGPMF – UNIFORM.



Calculates the natural logarithm of the probability density function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$f(x) = \log \frac{1}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability density.

Restriction: $lower \leq x \leq upper$

If the argument is out of range, a missing value is returned.

$lower$

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

$upper$

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability density of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPDF ("UNIFORM", -0.5, -1, 4);
  PUT s1=;
  s2 = LOGPDF ("UNIFORM", 3, -1, 4);
  PUT s2=;
  s3 = LOGPDF ("UNIFORM", -3, -1, 4);
  PUT s3=;
  s4 = LOGPDF ("UNIFORM", 7, -1, 4);
  PUT s4=;
RUN;
```

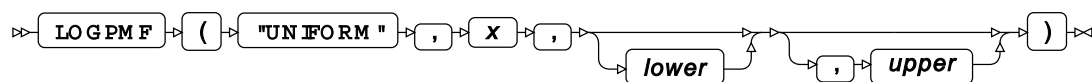
This produces the following output:

```
s1=-1.609437912
s2=-1.609437912
s3=.
s4=.
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

LOGPMF – UNIFORM

Returns the natural logarithm of the probability mass of the Uniform distribution at a specified point within the specified domain limits. This function is an alias of LOGPDF – UNIFORM.



Calculates the natural logarithm of the probability mass function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

$$f(x) = \log \frac{1}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the probability mass of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGPMF ("UNIFORM", -0.5, -1, 4);  
  PUT s1=;  
  s2 = LOGPMF ("UNIFORM", 3, -1, 4);  
  PUT s2=;  
  s3 = LOGPMF ("UNIFORM", -3, -1, 4);  
  PUT s3=;  
  s4 = LOGPMF ("UNIFORM", 7, -1, 4);  
  PUT s4=;  
RUN;
```

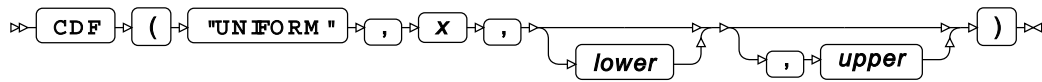
This produces the following output:

```
s1=-1.609437912  
s2=-1.609437912  
s3=.  
s4=.
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

CDF – UNIFORM

Returns the cumulative density of the Uniform distribution at a specified point within the specified domain limits.



Calculates the cumulative density function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x < lower & \text{return } 0 \\ \text{if } x > upper & \text{return } 1 \\ \text{otherwise} & \text{return } f(x) \end{cases}$$

$$f(x) = \frac{x - lower}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

$lower$

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

$upper$

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If $upper$ is specified, $lower$ must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the cumulative density of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = CDF("UNIFORM", -0.5, -1, 4);
  PUT s1;
  s2 = CDF("UNIFORM", 3, -1, 4);
  PUT s2;
  s3 = CDF("UNIFORM", -3, -1, 4);
  PUT s3;
  s4 = CDF("UNIFORM", 7, -1, 4);
  PUT s4;
RUN;
```

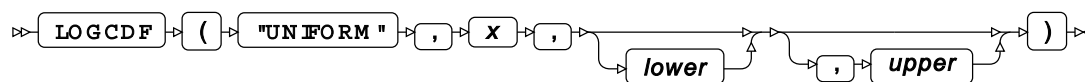
This produces the following output:

```
s1=0.1
s2=0.8
s3=0
s4=1
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

LOGCDF – UNIFORM

Returns the natural logarithm of the cumulative density of the Uniform distribution at a specified point within the specified domain limits.



Calculates the natural logarithm of the cumulative density function for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x > upper & \text{return } 0 \\ \text{if } upper \leq x \leq lower & \text{return } f(x) \end{cases}$$

$$f(x) = \log \frac{x - lower}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the cumulative density.

Restriction: $x \geq \text{lower}$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $\text{lower} < \text{upper}$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $\text{upper} > \text{lower}$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the cumulative density of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = LOGCDF ("UNIFORM", -0.5, -1, 4);  
  PUT s1=;  
  s2 = LOGCDF ("UNIFORM", 3, -1, 4);  
  PUT s2=;  
  s3 = LOGCDF ("UNIFORM", -3, -1, 4);  
  PUT s3=;  
  s4 = LOGCDF ("UNIFORM", 7, -1, 4);  
  PUT s4=;  
RUN;
```

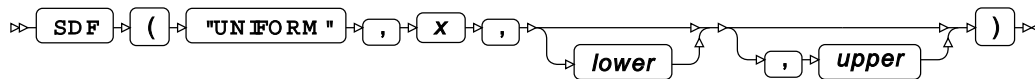
This produces the following output:

```
s1=-2.302585093  
s2=-0.223143551  
s3=.  
s4=0
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

SDF – UNIFORM

Returns the survival of the Uniform distribution at a specified point within the specified domain limits.



Calculates the survival, or the complement to the cumulative density function, for the Uniform distribution at point x . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x < lower & \text{return } 1 \\ \text{if } x > upper & \text{return } 0 \\ \text{otherwise} & \text{return } f(x) \end{cases}$$

$$f(x) = \frac{upper - x}{upper - lower}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the survival.

$lower$

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

$upper$

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: *upper* > *lower*

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the survival of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = SDF("UNIFORM", -0.5, -1, 4);
  PUT s1;
  s2 = SDF("UNIFORM", 3, -1, 4);
  PUT s2;
  s3 = SDF("UNIFORM", -3, -1, 4);
  PUT s3;
  s4 = SDF("UNIFORM", 7, -1, 4);
  PUT s4;
RUN;
```

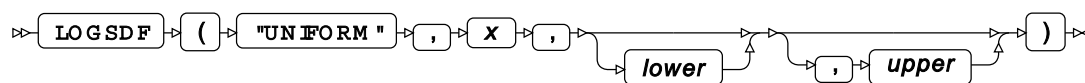
This produces the following output:

```
s1=0.9
s2=0.2
s3=1
s4=0
```

The first and second example show the output when *x* lies within the domain bounds. The third and fourth example show the output when *x* falls outside the domain bounds.

LOGSDF – UNIFORM

Returns the natural logarithm of the survival of the Uniform distribution at a specified point within the specified domain limits.



Calculates the natural logarithm of the survival, or the complement to the cumulative density function, for the Uniform distribution at point *x*. The last two arguments specify domain bounds for point *x*. If they are omitted, the following defaults are used: *lower* = 0 and *upper* = 1. These optional arguments must be either both specified or both omitted.

$$\begin{cases} \text{if } x < \text{lower} & \text{return } 0 \\ \text{if } \text{upper} \leq x \leq \text{lower} & \text{return } f(x) \end{cases}$$

$$f(x) = \log \frac{\text{upper} - x}{\text{upper} - \text{lower}}$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival.

Restriction: $x \leq upper$

If the argument is out of range, a missing value is returned.

lower

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If *lower* is specified, *upper* must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the survival of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGSDF("UNIFORM", 0.5, -1, 4);
  PUT s1=;
  s2 = LOGSDF("UNIFORM", 3, -1, 4);
  PUT s2=;
  s3 = LOGSDF("UNIFORM", -3, -1, 4);
  PUT s3=;
  s4 = LOGSDF("UNIFORM", 7, -1, 4);
  PUT s4=;
RUN;
```

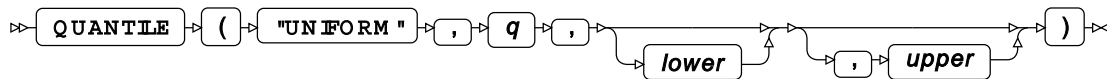
This produces the following output:

```
s1=-0.105360516
s2=-1.609437912
s3=0
s4=.
```

The first and second example show the output when x lies within the domain bounds. The third and fourth example show the output when x falls outside the domain bounds.

QUANTILE – UNIFORM

Returns the quantile of the Uniform distribution for a specified probability value within the specified domain limits.



Calculates the quantile x , or the inverse of the cumulative density function, for the Uniform distribution for probability value q . The last two arguments specify domain bounds for point x . If they are omitted, the following defaults are used: $lower = 0$ and $upper = 1$. These optional arguments must be either both specified or both omitted.

The quantile function returns point x such that randomly drawn values from the distribution fall below x with probability q .

$$f(q) = lower + q(upper - lower)$$

Return type: Numeric

q

Type: Numeric

The probability value for which the quantile is to be calculated.

Restriction: $0 < q < 1$

If the argument is out of range or contains a missing value, a missing value is returned.

$lower$

Optional argument

Type: Numeric

The lower domain bound for point x .

Default: 0

Restriction: $lower < upper$

If $lower$ is specified, $upper$ must also be specified.

If the argument is out of range, a missing value is returned.

upper

Optional argument

Type: Numeric

The upper domain bound for point x .

Default: 1

Restriction: $upper > lower$

If *upper* is specified, *lower* must also be specified.

If the argument is out of range, a missing value is returned.

Examples

In these examples, the quantile of the Uniform distribution is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = QUANTILE("UNIFORM", 0.1, -1, 4);  
  PUT s1=;  
  s2 = QUANTILE("UNIFORM", 0.8, -1, 4);  
  PUT s2=;  
  s3 = QUANTILE("UNIFORM", 0, -1, 4);  
  PUT s3=;  
  s4 = QUANTILE("UNIFORM", 1, -1, 4);  
  PUT s4=;  
RUN;
```

This produces the following output:

```
s1=-0.5  
s2=3  
s3=.  
s4=.
```

The first and second example show the output when q lies within the domain bounds. The third and fourth example show the output when q falls outside the domain bounds.

RAND – UNIFORM

Returns a random number from the Uniform distribution. This function is similar to RANUNI, UNIFORM and CALL RANUNI.

⇒ **RAND** ⇒ (⇒ **"UNIFORM"** ⇒) ⇒

This function does not take any variable arguments.

Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [↗](#) (page 777) before this function.

Return type: Numeric

The return value is between 0 and 1, not including the bounds.

Example

In this example, a random number from the Uniform distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("UNIFORM");
    PUT result;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.954524576
0.5654623868
0.0192114085
0.6654482979
0.5257064517
```

Running the `DATA` step again produces the following output.

```
The random numbers are:
0.5894905933
0.3842783501
0.2755029649
0.2809083956
0.8773915851
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the `DATA` step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(9);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("UNIFORM");
    PUT result;
  END;
RUN;
```

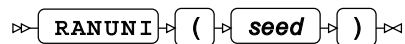

This produces the following output:

```
The random numbers are:  
0.7122548345  
0.2568146291  
0.864003316  
0.6932850354  
0.0290610364
```

Running the `DATA` step again produces the same output.

RANUNI

Returns a random number from the Uniform distribution. This function is an alias of `UNIFORM`. This function is similar to `RAND – UNIFORM` and `CALL RANUNI`.



```
RANUNI ( seed )
```

The first time you execute this function within a `DATA` step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a `DATA` step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is between 0 and 1, not including the bounds.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

Example

In this example, a random number from the Uniform distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result=ranuni(4);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.7398502793  
0.8803548626  
0.5992972244  
0.0375945815  
0.6864103115
```

Running the DATA step again produces the same output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;  
  PUT "The random numbers are:";  
  DO i = 1 TO 5;  
    result=ranuni(0);  
    PUT result;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.7800924917  
0.3883556134  
0.5708876502  
0.858912955  
0.1047348027
```

Running the DATA step again produces the following output.

```
The random numbers are:  
0.4455223314  
0.9903660994  
0.239625964  
0.9134768769  
0.2687704215
```

UNIFORM

Returns a random number from the Uniform distribution. This function is an alias of RANUNI. This function is similar to RAND – UNIFORM and CALL RANUNI.

➤ **UNIFORM** ➤ (➤ *seed* ➤) ➤

The first time you execute this function within a **DATA** step, the stream of random numbers is initialised *seed*; in all subsequent calls to this function *seed* is ignored. To generate the same sequence of random numbers each time the **DATA** step is executed, set *seed* to a negative value or zero. To generate a different sequence of random numbers each time the **DATA** step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this function, a new random number is generated; this includes the first use of this function within a **DATA** step. The first random number is returned immediately after the random stream has been initialised.

Return type: Numeric

The return value is between 0 and 1, not including the bounds.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

CALL RANUNI

Returns a random number from the Uniform distribution. This routine is similar to function RAND – UNIFORM, UNIFORM and RANUNI.

➤ **CALL RANUNI** ➤ (➤ *seed* ➤ , ➤ *x* ➤) ➤ ; ➤

Important:

The argument *seed* must be specified as a variable. If you specify a literal number here, the routine might return invalid results.

The first time you execute this routine within a **DATA** step, the stream of random numbers is initialised with the specified *seed*. Every time you update the *seed*, the stream is re-initialised.

To generate the same sequence of random numbers each time the `DATA` step is executed, set *seed* to a negative value or zero. This enables you to generate several reproducible sequences of random numbers from the same `DATA` step. To generate a different sequence of random numbers each time the `DATA` step is executed, set *seed* to a positive value greater than or equal to 1. If the value specified for *seed* is fractional, it is truncated to the nearest integer.

The random numbers are generated using the linear congruential generator. Each time you execute this routine, a new random number is generated; this includes each use with an updated *seed*. The random number is generated immediately after the stream has been initialised.

The return value is between 0 and 1, not including the bounds.

seed

Type: Numeric

The number used to initialise the random number generator.

If the argument contains a missing value, a missing value is returned.

x

Type: Numeric

The argument into which the random number is returned.

Example

In this example, a random number from the Uniform distribution is returned on each iteration of the loop and stored in *ranN*. The results are written to the log.

```
DATA _NULL_;  
  DO i = 1 TO 5;  
    call ranuni(4, ranN);  
    PUT ranN;  
  END;  
RUN;
```

This produces the following output:

```
The random numbers are:  
0.7398502793  
0.8803548626  
0.5992972244  
0.0375945815  
0.6864103115
```

Running the `DATA` step again produces the following output.

If the initial seed is set to zero, each run of the DATA step produces a different sequence of random numbers. For example:

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    call ranuni(0, ranN);
    PUT ranN;
  END;
RUN;
```

This produces the following output:

```
The random numbers are:
0.2629459092
0.6250730639
0.047694399
0.561057254
0.2579885285
```

Running the DATA step again produces the following output.

```
The random numbers are:
0.9078109664
0.448588825
0.8284740643
0.1039238833
0.9260302335
```

Wald distribution

Functions for the Wald distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – WALD [↗](#)..... 1337

$$\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right)$$

Returns the value of the probability density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of PMF – WALD, PDF – IGAUSS and PMF – IGAUSS.

PMF – WALD [↗](#)..... 1339

$$\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)$$

Returns the value of the probability mass function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of PDF – WALD, PDF – IGAUSS and PMF – IGAUSS.

LOGPDF – WALD [↗](#)..... 1341

$$\log\left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)\right]$$

Returns the value of the natural logarithm of the probability density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGPMF – WALD, LOGPDF – IGAUSS and LOGPMF – IGAUSS.

LOGPMF – WALD [↗](#)..... 1344

$$\log\left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)\right]$$

Returns the value of the natural logarithm of the probability mass function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGPDF – WALD, LOGPDF – IGAUSS and LOGPMF – IGAUSS.

CDF – WALD [↗](#)..... 1346

$$\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(\frac{-\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Returns the value of the cumulative density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of CDF – IGAUSS.

LOGCDF – WALD [↗](#)..... 1349

$$\log\left[\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(\frac{-\lambda(t-\mu)^2}{2\mu^2 t}\right) dt\right]$$

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGCDF – IGAUSS.

SDF – WALD [↗](#)..... 1351

$$1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(\frac{-\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Returns the value of the survival function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of SDF – IGAUSS.

LOGSDF – WALD [↗](#)..... 1354

$$\log\left[1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(\frac{-\lambda(t-\mu)^2}{2\mu^2 t}\right) dt\right]$$

Returns the value of the natural logarithm of the survival function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGSDF – IGAUSS.

QUANTILE – WALD [↗](#)..... 1356
 $\inf \{x: q \leq \text{CDF}(x; \lambda, \mu)\}$

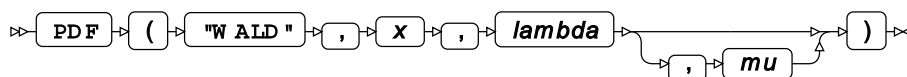
Returns the value of the quantile function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of QUANTILE – IGAUSS.

DEVIANCE – WALD [↗](#)..... 1359
 $(x - \mu)^2 / x\mu^2$

Returns the deviance of the Wald distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – IGAUSS.

PDF – WALD

Returns the value of the probability density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of PMF – WALD, PDF – IGAUSS and PMF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability density.

lambda

Type: Numeric

Restriction: *lambda* > 0

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = PDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = PDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = PDF("WALD", 3.5, 2, 1);
PUT s4=;

s5 = PDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.6429310692
s2=0.0562223942
s3=0.9678828981
s4=0.0144476475
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, PDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = PDF("WALD", 0.5);
PUT s2=;

s3 = PDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = PDF("WALD", 0.5, 1, 0);
PUT s4=;

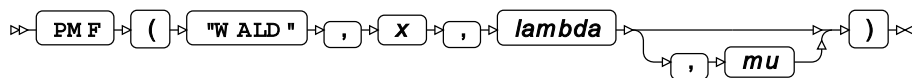
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

PMF – WALD

Returns the value of the probability mass function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of PDF – WALD, PDF – IGAUSS and PMF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the probability mass.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, PMF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = PMF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = PMF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = PMF("WALD", 3.5, 2, 1);
PUT s4=;

s5 = PMF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.6429310692
s2=0.0562223942
s3=0.9678828981
s4=0.0144476475
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, PMF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = PMF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = PMF("WALD", 0.5);
PUT s2=;

s3 = PMF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = PMF("WALD", 0.5, 1, 0);
PUT s4=;

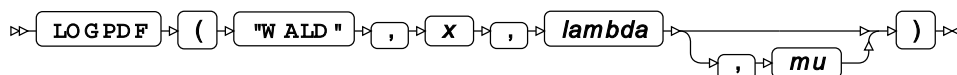
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGPDF – WALD

Returns the value of the natural logarithm of the probability density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGPMF – WALD, LOGPDF – IGAUSS and LOGPMF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point x is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(-\frac{\lambda(x-\mu)^2}{2\mu^2 x}\right) \right]$$

Return type: Numeric

x

Type: Numeric

Restriction: $x > 0$

The point at which to calculate the natural logarithm of the probability density.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = LOGPDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = LOGPDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = LOGPDF("WALD", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-0.441717762
s2=-2.878440129
s3=-0.032644172
s4=-4.237223681
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGPDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;
s1 = LOGPDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = LOGPDF("WALD", 0.5);
PUT s2=;

s3 = LOGPDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = LOGPDF("WALD", 0.5, 1, 0);
PUT s4=;

s5 = LOGPDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

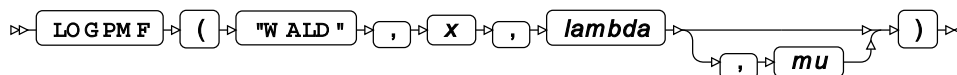
```
s1=.
s2=.
s3=.
s4=.
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

LOGPMF – WALD

Returns the value of the natural logarithm of the probability mass function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGPDF – WALD, LOGPDF – IGAUSS and LOGPMF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right) \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the probability mass.

If the argument is out of range, a missing value is returned.

lambda**Type:** Numeric**Restriction:** $x > 0$ ***mu***

Optional argument

Type: Numeric**Default:** 1**Restriction:** $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGPMF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = LOGPMF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = LOGPMF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = LOGPMF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = LOGPMF("WALD", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-0.441717762
s2=-2.878440129
s3=-0.032644172
s4=-4.237223681
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGPMF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;
  s1 = LOGPMF("WALD", 0.5, 1, 2, 3);
  PUT s1=;

  s2 = LOGPMF("WALD", 0.5);
  PUT s2=;

  s3 = LOGPMF("WALD", 0.5, 0, 2);
  PUT s3=;

  s4 = LOGPMF("WALD", 0.5, 1, 0);
  PUT s4=;

  s5 = LOGPMF("WALD", -0.5, 1, 2);
  PUT s5=;

RUN;
```

This produces the following output:

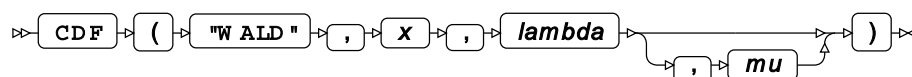
```
s1=.
s2=.
s3=.
s4=.
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

CDF – WALD

Returns the value of the cumulative density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of CDF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point x is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$
$$f(x; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the cumulative density.

λ

Type: Numeric

Restriction: $\lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

μ

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, CDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = CDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = CDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = CDF("WALD", 3.5, 2, 1);
PUT s4=;

s5 = CDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.2492117733
s2=0.8481757552
s3=0.2323571892
s4=0.9888921338
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, CDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = CDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = CDF("WALD", 0.5);
PUT s2=;

s3 = CDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = CDF("WALD", 0.5, 1, 0);
PUT s4=;

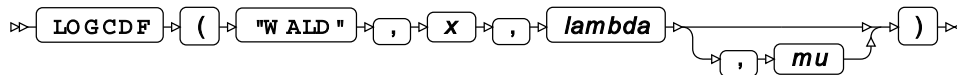
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGCDF – WALD

Returns the value of the natural logarithm of the cumulative density function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGCDF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for:

$$\begin{cases} x > 0 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$f(x; \lambda, \mu) = \log \left[\sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt \right]$$

Return type: Numeric

x

Type: Numeric

Restriction: $x > 0$

The point at which to calculate the natural logarithm of the cumulative density.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGCDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = LOGCDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = LOGCDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = LOGCDF("WALD", 3.5, 2, 1);
PUT s4=;

RUN;
```

This produces the following output:

```
s1=-1.389452249
s2=-0.164667406
s3=-1.459479483
s4=-0.011170019
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1.

Argument errors

In this example, LOGCDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGCDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = LOGCDF("WALD", 0.5);
PUT s2=;

s3 = LOGCDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = LOGCDF("WALD", 0.5, 1, 0);
PUT s4=;

s5 = LOGCDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

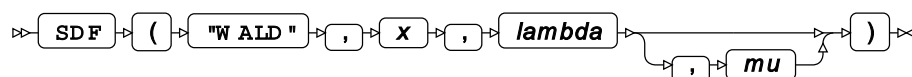
```
s1=.
s2=.
s3=.
s4=.
s5=M
```

The first four examples generate a message in the log, and return a missing value. The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth example generates a message in the log and returns MISSING_M, the missing value corresponding to $-\infty$. This example specifies an invalid value for *x*, which results in an attempt to calculate the natural logarithm of 0 (zero).

SDF – WALD

Returns the value of the survival function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of SDF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 1 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$
$$f(x; \lambda, \mu) = 1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(-\frac{\lambda(t-\mu)^2}{2\mu^2 t}\right) dt$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the value of the survival function.

lambda

Type: Numeric

Restriction: *lambda* > 0

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: *mu* > 0

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, SDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = SDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = SDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = SDF("WALD", 3.5, 2, 1);
PUT s4=;

s5 = SDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=0.7507882267
s2=0.1518242448
s3=0.7676428108
s4=0.0111078662
s5=1
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, SDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = SDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = SDF("WALD", 0.5);
PUT s2=;

s3 = SDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = SDF("WALD", 0.5, 1, 0);
PUT s4=;

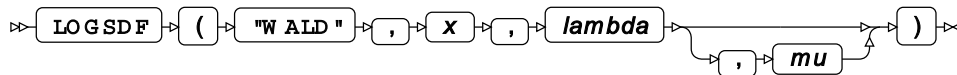
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

LOGSDF – WALD

Returns the value of the natural logarithm of the survival function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of LOGSDF – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *x* is

$$\begin{cases} \text{if } x \leq 0 & \text{return } 0 \\ \text{if } x > 0 & \text{return } f(x; \lambda, \mu) \end{cases}$$

$$f(x; \lambda, \mu) = \log \left[1 - \sqrt{\frac{\lambda}{2\pi}} \int_0^x \sqrt{\frac{1}{t^3}} \exp\left(\frac{-\lambda(t-\mu)^2}{2\mu^2 t}\right) dt \right]$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm of the survival function.

lambda

Type: Numeric

Restriction: *lambda* > 0

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $\mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, LOGSDF – WALD is called for distributions with various distribution parameters. The results are written to the log.

```
DATA _NULL_;

s1 = LOGSDF("WALD", 0.5, 1, 2);
PUT s1=;
s2 = LOGSDF("WALD", 3.5, 1, 2);
PUT s2=;

s3 = LOGSDF("WALD", 0.5, 2, 1);
PUT s3=;
s4 = LOGSDF("WALD", 3.5, 2, 1);
PUT s4=;

s5 = LOGSDF("WALD", -0.5, 1, 2);
PUT s5=;

RUN;
```

This produces the following output:

```
s1=-0.286631655
s2=-1.885031712
s3=-0.264430744
s4=-4.500101754
s5=0
```

The first two examples specify two points in a distribution with a shape parameter (*lambda*) of 1 and a mean (*mu*) of 2. The third and fourth examples specify the same two points in a distribution with shape parameter 2 and mean 1. The fifth example specifies a negative value for *x*.

Argument errors

In this example, LOGSDF – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

s1 = LOGSDF("WALD", 0.5, 1, 2, 3);
PUT s1=;

s2 = LOGSDF("WALD", 0.5);
PUT s2=;

s3 = LOGSDF("WALD", 0.5, 0, 2);
PUT s3=;

s4 = LOGSDF("WALD", 0.5, 1, 0);
PUT s4=;

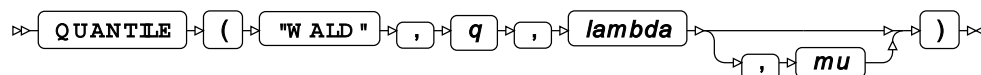
RUN;
```

All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

QUANTILE – WALD

Returns the value of the quantile function at a specified point for the Wald distribution with the specified shape and mean. This function is an alias of QUANTILE – IGAUSS.



The Wald distribution is also known as the Inverse Gaussian distribution.

The distribution parameters are λ (*lambda*), the shape, and μ (*mu*), the mean. *mu* is optional. If *mu* is not specified, *mu* defaults to 1.

This function is defined for

$$\begin{cases} 0 < q < 1 \\ \lambda > 0 \\ \mu > 0 \end{cases}$$

The calculated value at point *q* is

$$f(q; \lambda, \mu) = \inf \{x: q \leq \text{CDF}(x; \lambda, \mu)\}$$

where

- $x \in \mathbb{R}$

- $\inf\{x\}$ (*infinium*) is the greatest lower bound of x
- λ is the shape parameter
- μ is the mean
- CDF ($x; \lambda, \mu$) is the cumulative density function

Return type: Numeric

q

Type: Numeric

Restriction: $0.0 < q < 1.0$

The probability value for which to calculate the quantile.

If the argument is out of range, a missing value is returned.

lambda

Type: Numeric

Restriction: $lambda > 0$

The shape parameter.

If the argument is out of range, a missing value is returned.

mu

Optional argument

Type: Numeric

Default: 1

Restriction: $mu > 0$

The mean of the distribution.

If the argument is out of range, a missing value is returned.

Basic example

In this example, QUANTILE – WALD is called twice, with two different probability values from the same distribution. The returned values are then passed to CDF – WALD for the same distribution. The results are written to the log.

```
DATA _NULL_;

  s1 = QUANTILE("WALD", 0.25, 1, 2);
  PUT s1=;
  s2 = QUANTILE("WALD", 0.75, 1, 2);
  PUT s2=;

  s3 = CDF("WALD", s1, 1, 2);
  PUT s3=;
  s4 = CDF("WALD", s2, 1, 2);
  PUT s4=;

RUN;
```

This produces the following output:

```
s1=0.5012268362
s2=2.2841739815
s3=0.25
s4=0.75
```

As QUANTILE – WALD and CDF – WALD are inverse functions, the values returned from CDF – WALD are the same as those originally passed to QUANTILE – WALD.

Argument errors

In this example, QUANTILE – WALD is called with various combinations of invalid arguments. The results are written to the log.

```
DATA _NULL_;

  s1 = QUANTILE("WALD", 0.25, 1, 2, 3);
  PUT s1=;

  s2 = QUANTILE("WALD", 0.25);
  PUT s2=;

  s3 = QUANTILE("WALD", 0.25, 0, 2);
  PUT s3=;

  s4 = QUANTILE("WALD", 0.25, 1, 0);
  PUT s4=;

  s5 = QUANTILE("WALD", 0, 1, 0);
  PUT s5=;

  s6 = QUANTILE("WALD", -1, 1, 0);
  PUT s6=;

RUN;
```

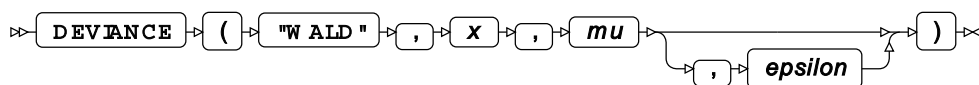
All these examples generate a message in the log, and return a missing value.

The first example has too many arguments. The second example has too few arguments. The third example specifies an invalid value for *lambda* and the fourth example specifies an invalid value for *mu*.

The fifth and sixth examples specify invalid values for *q*.

DEVIANCE – WALD

Returns the deviance of the Wald distribution at a specified point, based on the distribution mean. This function is an alias of DEVIANCE – IGAUSS.



Keyword `WALD` is an alias of `IGAUSS`, see [DEVIANCE – IGAUSS](#) (page 1040).

Calculates the deviance, or goodness of fit, for the generalised linear model of the Wald distribution at a nonnegative point x based on the distribution mean μ (mu). An optional range correction parameter ε (*epsilon*) can be specified. If $\varepsilon > 0.01$, it is set equal to 0.01. If it is not specified or if $\varepsilon < 10^{-12}$, the value of 10^{-12} is used for correction. The distribution mean is then adjusted so that $\mu \geq \varepsilon$:

$$\text{if } \mu < \varepsilon \text{ set } \mu = \varepsilon$$

If $x \geq 0$, it is adjusted so that $x \geq \varepsilon$:

$$\text{if } 0 \leq x < \varepsilon \text{ set } x = \varepsilon$$

These adjusted values of μ and x are used in the subsequent calculation of the deviance.

$$(x - \mu)^2 / x\mu^2$$

Return type: Numeric

x

Type: Numeric

The point at which to calculate the deviance.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

mu

Type: Numeric

The distribution mean.

Expected: $\mu > 0$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

epsilon

Optional argument

Type: Numeric

The range correction parameter.

Default: $\varepsilon = 10^{-12}$

Expected: $10^{-12} < \varepsilon < 0.01$. Values not within this range are corrected to fall in this range; this behaviour is however, deprecated, and might be removed in future.

If the argument contains a missing value, a missing value is returned.

Weibull distribution

Functions for the Weibull distribution.

Both the probability density function and the probability mass function are defined for this distribution. These functions return identical results.

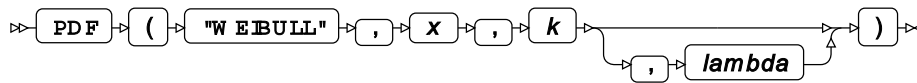
Probability functions are related to each other as follows:

- The cumulative density function is the cumulative version of the probability density function over the distribution domain.
- The survival function is the complement to the cumulative density function.
- The quantile function is the inverse of the cumulative density function.

PDF – WEIBULL ↗	1361
PMF – WEIBULL ↗	1361
LOGPDF – WEIBULL ↗	1362
LOGPMF – WEIBULL ↗	1362
CDF – WEIBULL ↗	1363
LOGCDF – WEIBULL ↗	1363
SDF – WEIBULL ↗	1364
LOGSDF – WEIBULL ↗	1364
QUANTILE – WEIBULL ↗	1365
RAND – WEIBULL ↗	1365

Returns a random number from the Weibull distribution based on the shape and scale.

PDF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

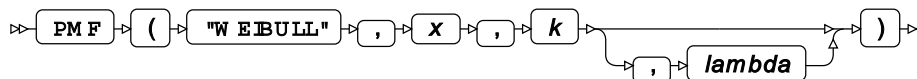
Type: Numeric

lambda

Optional argument

Type: Numeric

PMF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

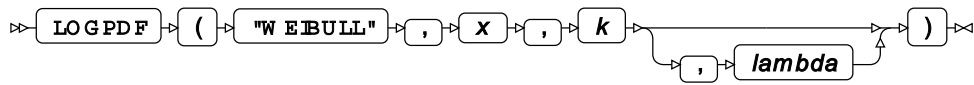
Type: Numeric

lambda

Optional argument

Type: Numeric

LOGPDF – WEIBULL



Return type: Numeric

x

Type: Numeric

 k

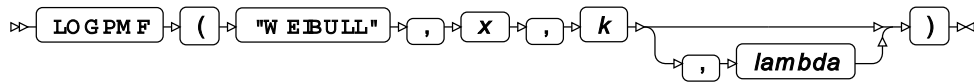
Type: Numeric

lambda

Optional argument

Type: Numeric

LOGPMF – WEIBULL



Return type: Numeric

X

Type: Numeric

 \mathbf{k}

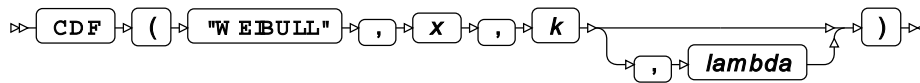
Type: Numeric

lambda

Optional argument

Type: Numeric

CDF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

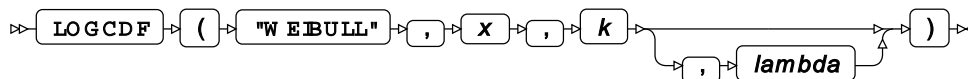
Type: Numeric

lambda

Optional argument

Type: Numeric

LOGCDF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

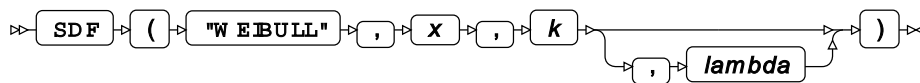
Type: Numeric

lambda

Optional argument

Type: Numeric

SDF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

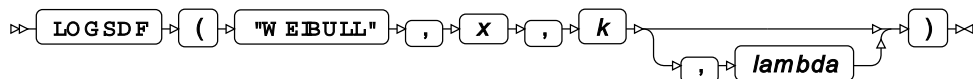
Type: Numeric

lambda

Optional argument

Type: Numeric

LOGSDF – WEIBULL



Return type: Numeric

x

Type: Numeric

k

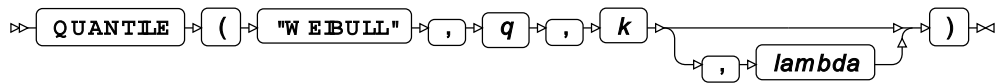
Type: Numeric

lambda

Optional argument

Type: Numeric

QUANTILE – WEIBULL



Return type: Numeric

q

Type: Numeric

k

Type: Numeric

lambda

Optional argument

Type: Numeric

RAND – WEIBULL

Returns a random number from the Weibull distribution based on the shape and scale.



Each time you execute this function within a `DATA` step, a new random number is generated using the Mersenne Twister algorithm, see: Makoto Matsumoto and Takuji Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation (TOMACS)* - Special issue on uniform random number generation 8, no. 1 (1998), 3-30.

If the random stream is not initialised, repeated executions of the same `DATA` step produce different sequences of random numbers. To initialise the random stream, use `CALL STREAMINIT` [\(page 777\)](#) before this function.

Return type: Numeric

The return value is positive.

shape

Type: Numeric

The shape of the distribution.

scale**Type:** Numeric

The scale of the distribution.

Example

In this example, a random number from the Weibull distribution is returned on each iteration of the loop. The results are written to the log.

```
DATA _NULL_;
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("WEIBULL", 5,20);
    PUT result;
  END;
run
```

This produces the following output:

```
The random numbers are:
11.45651107
10.633667085
19.256692236
14.987748594
11.624668812
```

Running the DATA step again produces the following output.

```
The random numbers are:
14.192837117
21.534350475
19.95207052
20.571542546
18.188153803
```

However, if you first use `CALL STREAMINIT` to specify a seed, then each time you run the DATA step, it produces the same output.

```
DATA _NULL_;
  CALL STREAMINIT(9);
  PUT "The random numbers are:";
  DO i = 1 TO 5;
    result = RAND("WEIBULL", 5,20);
    PUT result;
  END;
RUN;;
```

This produces the following output:

```
The random numbers are:
1.7506445418
0.5203794406
2.9996140074
1.6572896735
0.1503519974
```

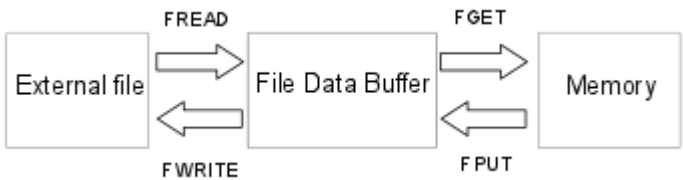
Running the `DATA` step again produces the same output.

External file functions

Open and close files that have formats external to WPS, and perform other operations using those files.

WPS and the SAS language create particular kinds of files in which they store datasets. These datasets are the default type of files written and read. Other file types, known as external files, can also be read. For example, a Windows text file, or a comma-separated values file can be opened and read and written to.

When external files are read, records are stored in the File Data Buffer (FDB), which acts as a staging area between the WPS environment and external files. Records can be obtained from the FDB and placed in variables. The information in those variables can then be written back to external files, to datasets, or manipulated in the `DATA` step. The figure below shows the flow of data through the FDB, and the four external file functions used to read and write FDB records.



`FREAD` reads a record from the external file, and puts it into the FDB. You then use `FGET` to obtain the record from the FDB and store it in a variable. `FPUT` puts the data in a variable into the FDB. `FWRITE` then writes the data to the external file.

You can perform operations on both folders and files. A folder or file must be first be opened before use, which creates an identifier that is then used by other functions to identify that file or folder. The folder or file should be closed after use; this also releases the identifier for use in subsequent open operations.

<code>DCLOSE</code> ↗	1369
Closes a folder opened with the <code>DOPEN</code> function, and releases the identifier associated with it.	
<code>DCREATE</code> ↗	1370
Creates a directory or folder in a specified folder.	
<code>DINFO</code> ↗	1372
Returns a specified property of the specified folder or directory.	
<code>DNUM</code> ↗	1373
Returns the number of items in a folder.	
<code>DOPEN</code> ↗	1374
Returns an identifier for a specified directory or folder that can then be used in other folder functions.	
<code>DOPTNAME</code> ↗	1375
Returns a specified option or property for an open folder.	

DOPTNUM ↗	1376
Returns the number of properties associated with a specified folder.	
DREAD ↗	1377
Returns the name of an item in a folder at a specified index position.	
DROPNOTE ↗	1378
Drops a note previously created using <code>FNOTE</code> , and releases the note identifier.	
DSNCATLGD ↗	1379
Returns a value that indicates whether the specified dataset is cataloged. (z/OS only).	
FCLOSE ↗	1380
Closes a file opened with the <code>FOPEN</code> function, and release the identifier associated with it.	
FCOL ↗	1380
Returns the character (column) position of the file pointer in the current record in the File Data Buffer.	
FDELETE ↗	1382
Deletes the specified file.	
FEXIST ↗	1383
Returns a value that indicates whether a fileref exists, where that fileref has been created using <code>FILENAME</code> .	
FGET ↗	1383
Gets a record from the File Data Buffer and puts it into memory.	
FILEEXIST ↗	1388
Returns a value that indicates whether a specified file exists, based on its physical location.	
FILENAME ↗	1389
Creates a reference to a file or fileref that can be used in other external file functions, such as <code>FOPTNAME</code> and <code>FINFO</code>	
FILEREf ↗	1390
Returns a value indicating whether a specified file reference has been assigned.	
FINFO ↗	1391
Returns the value of a specified property of a file.	
FNOTE ↗	1392
Returns an identifier for a record in the File Data Buffer. The record can then be located using <code>FPOINT</code> and the identifier.	
FOPEN ↗	1393
Returns an identifier for a specified file that can then be used in other file functions to specify the file.	
FOPTNAME ↗	1396
Returns a specified option or property for an open file.	
FOPTNUM ↗	1397
Returns the number of properties associated with a specified file.	

FPOINT ↗	1398
Points the file pointer at a record in the File Data Buffer specified by a note identifier. A note identifier is defined using <code>FNOTE</code> .	
FPOS ↗	1399
Positions the file pointer at the specified column in the current row in the file.	
FPUT ↗	1401
Puts (writes) a record into the File Data Buffer (FDB).	
FREAD ↗	1404
Reads a record from a specified file to the File Data Buffer (FDB).	
FRECCNT ↗	1405
Returns the number of records in a file opened with <code>FOPEN</code> .	
FREWIND ↗	1405
Positions the file pointer before the first record.	
FRLen ↗	1408
Returns the length of the current record.	
FSEP ↗	1409
Specifies the separator for the record obtained by an <code>FREAD</code> function.	
FWRITE ↗	1410
Writes a record from the File Data Buffer to a specified file.	
MOPEN ↗	1413
Returns an identifier for a specified member or file that can then be used in other file functions to specify the member or file.	
PATHNAME ↗	1415
Returns the full pathname for a folder or filename that has already been specified using <code>FILENAME</code> or <code>LIBNAME</code> .	
SYSMSG ↗	1418
Returns an error message for the external file functions.	
SYSRC ↗	1418
Returns the error code for an external file function.	

DCLOSE

Closes a folder opened with the `DOPEN` function, and releases the identifier associated with it.

```
»» DCLOSE ( directory-id ) ««
```

The folder with the specified identifier is closed. An identifier is obtained using the `DOPEN` [↗](#) (page 1374) function. When the folder is closed, the identifier is released and becomes available to a subsequent `DOPEN` function.

Return type: Numeric

0 (zero) is always returned.

directory-id

Type: Numeric

The identifier of a folder previously opened using `DOPEN` [↗](#) (page 1374).

Example

In this example, the `DOPEN` function returns an identifier for the specified folder references. The references are created by the `FILENAME` function. The folders are closed and the identifiers deassigned using the `DCLOSE` function. The result is written to the log.

```
DATA _NULL_;

rc = FILENAME("mydir1", "C:\");
dir_id=DOPEN("mydir1");
PUT "The identifier is: " dir_id;
y = DCLOSE(dir_id);

rc = FILENAME("mydir2", "c:\temp");
dir_id=DOPEN("mydir2");
PUT "The identifier is: " dir_id;
y = DCLOSE(dir_id);

RUN;
```

This produces the following output:

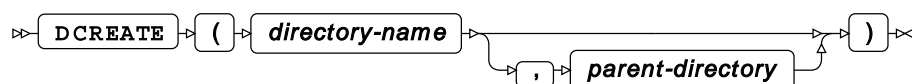
```
The identifier is: 1
The identifier is: 1
```

In this example:

- The first identifier returned is 1, because `C:\` is a valid pathname
- The second identifier returned is 1, because the pathname is valid, and the `DCLOSE` function was used before this `DOPEN` to close the previously opened folder and release the identifier

DCREATE

Creates a directory or folder in a specified folder.



Return type: Character

The pathname of the folder created.

directory-name**Type:** Character

The name of the folder to be created.

parent-directory

Optional argument

Type: Character

The folder in which you want to create the folder.

If *parent-directory* is not specified, the pathname of the folder is returned, but no folder is created.

Example

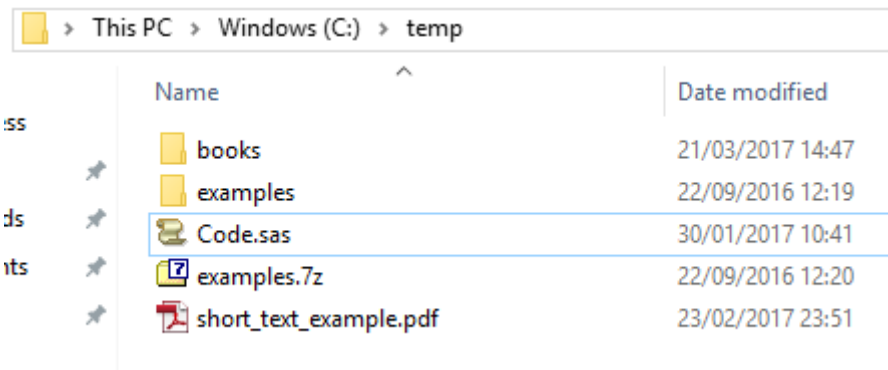
In this example, the function creates a new subfolder in the specified folder. The result is written to the log.

```
DATA _NULL_;  
  y = DCREATE("test", "c:\temp");  
  PUT y= ;  
RUN;
```

This produces the following output:

```
y=c:\temp\test
```

In this example, a new folder `test` is created in the folder `C:\temp`. The pathname of the new folder is saved in variable `y`, enabling you to specify this folder in other functions. In the figures below, you can see the list of items before the new folder is created, and after the folder is created, respectively.



	Name	Date modified
	books	21/03/2017 14:47
	examples	22/09/2016 12:19
	Code.sas	30/01/2017 10:41
	examples.7z	22/09/2016 12:20
	short_text_example.pdf	23/02/2017 23:51

Name	Date modified
books	21/03/2017 14:47
examples	22/09/2016 12:19
test	23/03/2017 15:17
Code.sas	30/01/2017 10:41
examples.7z	22/09/2016 12:20
short_text_example.pdf	23/02/2017 23:51

DINFO

Returns a specified property of the specified folder or directory.

```
DINFO ( directory-id , property )
```

Each folder or directory has a number of properties or options associated with it, such as the type, name, and so on. You can get a list of the properties using the [DOPTNUM](#) (page 1375) and [DOPTNAME](#) (page 1375) functions, and then use this function to get the value of a property in that list.

The properties available depend on the operating system.

The folder is specified using an identifier generated by [DOPEN](#) (page 1374).

Return type: Character

directory-id

Type: Numeric

The identifier of a folder previously opened using [DOPEN](#) (page 1374).

property

Type: Character

The name of a property returned by [DOPTNAME](#). The properties returned, and the values associated with them, depend on the operating system.

Example

In this example, the value of all properties for the specified folder are returned. The number of properties is first returned using [FOPTNUM](#), the names of the properties are returned using [FOPTNAME](#), and the values for each property are then returned using [FINFO](#). The result is written to the log.

Note:

On Windows operating systems, the only property available is the folder name.

```
DATA _NULL_;
  rc = FILENAME("mydir", "C:\temp");
  o = DOPEN("mydir");
  x = DOPTNUM(o);
  DO i = 1 TO x;
    y = DOPTNAME(o, i);
    n = DINFO(o, y);
    PUT "The value of the property " y "is: " n;
  END;
RUN;
```

This produces the following output:

```
The value of the property Directory is: C:\temp
```

DNUM

Returns the number of items in a folder.

➤ **DNUM** ➤ (➤ *directory-id* ➤) ➤

The items in a folder might be subfolders, files, shortcuts, and so on. The count does not include items in subfolders.

Return type: Numeric

directory-id

Type: Numeric

The identifier of a folder previously opened using `DOPEN` [↗](#) (page 1374).

Example

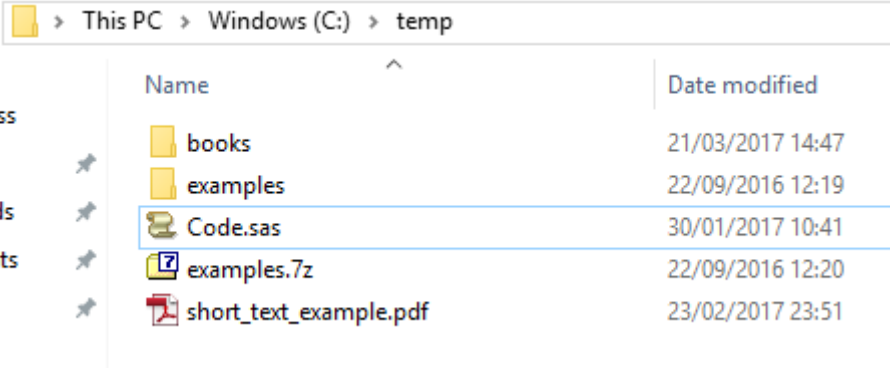
In this example, the number of items in the folder is returned. The folder is opened using `DOPEN`, which also returns an identifier for the specified folder reference. The reference is created by the `FILENAME` function. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("mydir", "C:\temp");
  dir_id = DOPEN("mydir");
  x = DNUM(dir_id);
  PUT "The number of items in the folder is: " x;
RUN;
```

This produces the following output:

```
The number of items in the folder is: 5
```

In this example, the contents of the folder are :



	Name	Date modified
ss	books	21/03/2017 14:47
ds	examples	22/09/2016 12:19
its	Code.sas	30/01/2017 10:41
	examples.7z	22/09/2016 12:20
	short_text_example.pdf	23/02/2017 23:51

As can be seen, the folder `C:\temp` does contain five items.

DOPEN

Returns an identifier for a specified directory or folder that can then be used in other folder functions.

➤ **DOPEN** ➤ (➤ *fileref* ➤) ➤

You cannot get an identifier for a specified folder directly. You must first use the `FILENAME` function to assign a folder reference, and then use `DOPEN` to specify an identifier for that reference.

Return type: Numeric

An integer greater than or equal to 0.

fileref

Type: Character

A name, assigned in the `FILENAME` function, that is the reference for the folder pathname.

If the folder does not exist, the identifier 0 is returned. If the folder exists, the identifier 1 is assigned to the first folder opened. If another folder is then opened, the identifier is set to 2. The identifier increments by one for each folder opened within the same `DATA` step, unless a previously opened folder is closed using `DCLOSE`, in which case the released identifier is used.

For Windows, the drive or partition letter (for example, `C:`) is not accepted as the root of the drive. Instead, you must specify the top-level directory, specified using the drive letter and backslash; for example, `C:\`.

Example

In this example, the function returns identifiers for the specified folder references. The references are specified using the `FILENAME` function. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("mydir", "C:");
  dir_id=DOPEN("mydir");
  PUT "The identifier is: " dir_id;
  rc = FILENAME("mydir2", "C:\");
  dir_id=DOPEN("mydir2");
  PUT "The identifier is: " dir_id;
  y = dclose(dir_id);
  rc = FILENAME("mydir3", "c:\temp");
  dir_id=DOPEN("mydir3");
  PUT "The identifier is: " dir_id;
  rc = FILENAME("mydir4", "m:\");
  dir_id=DOPEN("mydir4");
  PUT "The identifier is: " dir_id;
  rc = FILENAME("mydir5", "c:\temp\books");
  dir_id=DOPEN("mydir5");
  PUT "The identifier is: " dir_id;
RUN;
```

This produces the following output:

```
The identifier is: 0
The identifier is: 1
The identifier is: 1
The identifier is: 2
The identifier is: 3
```

In this example:

- The first identifier returned is 0, because `C:` is not a valid pathname.
- The second identifier returned is 1, because `C:\` is a valid pathname.
- The third identifier returned is 1, because the pathname is valid, and `DCLOSE` was used before this `DOPEN` to close the previously-opened directory and release the identifier.
- The fourth identifier returned is 2, because the pathname is valid, and the identifier has been incremented by one.
- The fifth identifier returned is 3, because the pathname is valid, and the identifier has been incremented by one.

DOPTNAME

Returns a specified option or property for an open folder.

```
DOPTNAME ( directory-id , index )
```

Each folder has a number of properties or options associated with it, such as the type, or the originating user, and so on. The properties or options are organised as a list. To find how many properties or options the folder has, use `DOPTNUM`, and then use this function to list a specified property or option at a specified position in the list.

The properties available depend on the operating system.

The folder is specified using an identifier generated by `DOPEN` [↗](#) (page 1374).

Return type: Character

directory-id

Type: Numeric

The identifier of a folder previously opened using `DOPEN` [↗](#) (page 1374).

index

Type: Numeric

The position of the property to return in the list of properties or options.

Example

In this example, the number of properties or options is returned by `DOPTNUM`, and then each property or option is returned using `DOPTNAME`. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("mydir", "C:\temp");
  dir_id=DOPEN("mydir");
  DO i = 1 TO DOPTNUM(dir_id);
    y = DOPTNAME(dir_id,i);
    PUT "property " i "is " y;
  END;
RUN;
```

This produces the following output:

```
property 1 is Directory
```

For this folder, there is only one property, `Directory`.

DOPTNUM

Returns the number of properties associated with a specified folder.

⇒ `DOPTNUM` ⇒ (⇒ *directory-id* ⇒) ⇒

Each folder has a number of properties or options associated with it, such as the type, or the originating user, and so on. You can find how many properties or options there are with this function, and then use `DOPTNAME` to list all the properties or options.

The properties available depend on the operating system.

The folder is specified using an identifier generated by `DOPEN` [\(page 1374\)](#).

Return type: Numeric

directory-id

Type: Numeric

The identifier of a folder previously opened using `DOPEN` [\(page 1374\)](#).

Example

In this example, the number of properties or options is returned by `DOPTNUM`, and then each property or option is returned using `DOPTNAME`. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("mydir","C:\temp");  
  dir_id=DOPEN("mydir");  
  DO i = 1 TO DOPTNUM(dir_id);  
    y = DOPTNAME(dir_id,i);  
    PUT "property " i "is " y;  
  END;  
RUN;
```

This produces the following output:

```
property 1 is Directory
```

For this folder, there is only one property, `Directory`.

DREAD

Returns the name of an item in a folder at a specified index position.

➤ **DREAD** ➤ (➤ *directory-id* ➤ , ➤ *index* ➤) ➤

Items in a folder, such as files and subfolders, are organised by this function into alphabetical order.

The function finds the item at the specified position in the list. For example, you can find the fourth item in a list of five items in a specified folder and return the name of that item.

The folder is specified using an identifier generated by `DOPEN`.

Return type: Character

directory-id

Type: Numeric

The identifier of a folder previously opened using `DOPEN` [↗](#) (page 1374).

index

Type: Numeric

The ordinal position in the list of items. For example, if you want to find the name of the fourth item in the list of items, enter 4.

Example

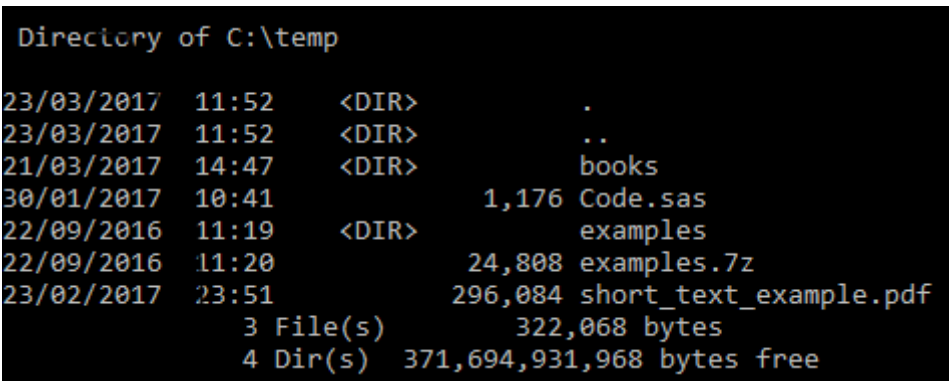
In this example, the name of the fourth item in a folder is returned. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("mydir","C:\temp");  
  dir_id = DOPEN("mydir");  
  x = DREAD(dir_id, 4);  
  PUT "The name of the specified item is: " x;  
RUN;
```

This produces the following output:

```
The name of the specified item is: examples.7z
```

In this example, the contents of the folder are:



```
Directory of C:\temp  
  
23/03/2017  11:52    <DIR>          .  
23/03/2017  11:52    <DIR>          ..  
21/03/2017  14:47    <DIR>          books  
30/01/2017  10:41                1,176 Code.sas  
22/09/2016  11:19    <DIR>          examples  
22/09/2016  11:20                24,808 examples.7z  
23/02/2017  23:51                296,084 short_text_example.pdf  
                3 File(s)                322,068 bytes  
                4 Dir(s)  371,694,931,968 bytes free
```

As can be seen, the fourth item in folder `C:\temp` is `examples.7z`.

DROPNOTE

Drops a note previously created using `FNOTE`, and releases the note identifier.

```
➤ DROPNOTE ➤ ( ➤ dataset-or-file-id ➤ , ➤ note-id ➤ ) ➤
```


Return type: Numeric

dataset-or-file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

note-id

Type: Numeric

A note identifier, created using `FNOTE` [↗](#) (page 1392).

Example

In this example, the specified file is read from beginning to end; the 1000th character is written to the log and noted using `FNOTE`. The file pointer is then repositioned using `FPOINT`, where the record is obtained and written to the log. The note is then dropped, and the note identifier released, using `DROPNOTE`.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile");
  rec = QUOTE("r");
  z = 1;
  IF f_id1 = 1;
    DO UNTIL (z = 2010);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rec);
      IF z = 1000 THEN ni = FNOTE(f_id1);
      IF z = 1000 THEN PUT rec=;
      z = z + 1;
    END;
    rc = FPOINT(f_id1, ni);
    rr = FREAD(f_id1);
    rc = FGET(f_id1, rec);
    PUT rec=;
    rc = DROPNOTE(f_id1, ni);
    rc = FCLOSE(f_id1);
  RUN;
```

This produces the following output:

```
rec=Breakfast
rec=Breakfast
```

DSNCATLGD

Returns a value that indicates whether the specified dataset is cataloged. (z/OS only).

➤ **DSNCATLGD** ➤ (➤ ***dataset-name*** ➤) ➤

Only available in WPS on the z/OS operating system.

Return type: Numeric

1 if the dataset is cataloged; 0 otherwise.

dataset-name

Type: Character

The name of the dataset to be checked.

FCLOSE

Closes a file opened with the `FOPEN` function, and release the identifier associated with it.

⇒ `FCLOSE` ⇒ (⇒ *file-id* ⇒) ⇒

The file with the specified identifier is closed. The identifier is created by the `FOPEN` function. The identifier is released, and becomes available to a subsequent `FOPEN` function.

Return type: Numeric

0 (zero) is always returned.

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, a file is opened and then closed.

```
DATA _NULL_;  
  rc = FILENAME("myfile", "C:\temp\examples.7z");  
  f_id=FOPEN("myfile");  
  rc = FCLOSE(f_id);  
RUN;
```

FCOL

Returns the character (column) position of the file pointer in the current record in the File Data Buffer.

⇒ `FCOL` ⇒ (⇒ *file-id* ⇒) ⇒

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, a string is written to the file at position 10 in the record. The `FCOL` function then returns the position of the pointer in the record after this operation. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\example.txt");
  f_id1 = FOPEN("myfile", "o");
  wrrec = "A record for the file";
  rc = FPOS(f_id1,10 );
  rc = FPUT(f_id1, wrrec);
  x = FCOL(f_id1);
  PUT "The pointer is at character position " x "in the record";
  rc = FWRITE(f_id1);
  RUN;
```

This produces the following output:

```
The pointer is at character position 31 in the record
```

In this example, the `FPOS` function first positions the pointer at position 10 in the record in the File Data Buffer (FDB). The `FPUT` function then puts the string into the FDB at that position. The string is 21 characters long. When `FPUT` finishes putting the string in the FDB, the pointer is at the end of the record, which is, therefore, position 31.

The value returned by the function depends on the operations that have occurred before it is executed. For example, if the `FCOL` statement and `PUT` in the previous example are moved to after the `FWRITE` function:

```
rc = FPOS(f_id1,10 );
rc = FPUT(f_id1, wrrec);
rc = FWRITE(f_id1);
x = FCOL(f_id1);
PUT "The pointer is at character position " x "in the record";
```

then the following is written to the log:

```
The pointer is at character position 1 in the record
```

Because the `FWRITE` function has now written the record from the FDB to the file, the current record is empty and the pointer is pointing at character position 1.

FDELETE

Deletes the specified file.



Return type: Numeric

If the specified file or folder is successfully deleted, 0 (zero) is returned; if it is not, the code 20004 is returned.

fileref-or-directory

Type: Character

A name, assigned by the `FILENAME` function, that is the reference for the pathname or file to be deleted.

If you are running WPS on z/OS, this function can only delete empty partitioned datasets (PDS or PDSE).

Example

In this example, the function deletes the specified files. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\test");
  rc = FDELETE("myfile");
  PUT rc=;
  rc = FILENAME("myfile", "C:\temp\test_1.txt");
  rc = FDELETE("myfile");
  PUT rc=;
RUN;
```

The folder `C:\temp\test` and the file `test_1.txt` in the folder `C:\temp` are deleted. The following values are returned, because the files were successfully deleted:

```
rc=0
rc=0
```

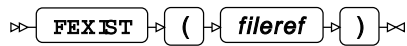
If you were to run the same DATA step again, the values returned would be:

```
rc=20004
rc=20004
```

because the files have already been deleted and so cannot be found.

FEXIST

Returns a value that indicates whether a fileref exists, where that fileref has been created using `FILENAME`.



1 if the file exists; 0 otherwise.

Return type: Numeric

1 if the file exists; 0 otherwise.

fileref

Type: Character

A variable containing the pathname and filename, specified using `FILENAME`.

Example

In this example, a fileref is specified using the `FILENAME` function. The `FEXIST` function is then used to check that the file corresponding to that fileref exists, and returns a value indicating whether it does. That value is used to generate a message using `IFC` (page 671). The result is written to the log.

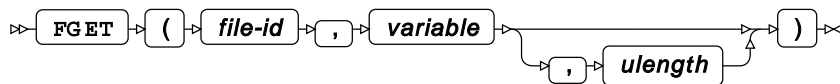
```
DATA _NULL_;  
  rc = FILENAME("myfile" , "C:\temp\books\books.accdb");  
  rc = FEXIST("myfile");  
  yn = IFC(rc, "The file exists", "The file does not exist");  
  PUT yn;  
RUN;
```

This produces the following output:

```
The file exists
```

FGET

Gets a record from the File Data Buffer and puts it into memory.



Records in the File Data Buffer (FDB) are placed there after being read from a file by `FREAD`.

Return type: Numeric

1 if the record exists, 0 (zero) otherwise.

file-id**Type:** NumericThe identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).**variable****Type:** Character

The name of the variable that contains the data to be obtained from the FDB.

ulength

Optional argument

Type: Numeric

The number of characters to read from the record in the FDB.

If you specify *ulength*, the length of the record returned might be shorter than the specified length, because a separator character might first occur.

variable must contain a string. It can be a quoted string you enter, for example, `s1 = "xxxxxxx"`, or a string you create in another function, for example, `gs = QUOTE("r")`.

The way in which you define the string has affects the way the data is obtained from the FDB. If you enter a string for *value* (for example, `s1 = "xxxxxxx"`), this becomes a mask, starting at position 1 of the record, for the contents of the record. For example, if you specify *value* as the string "record":

```
s1 = "xxxxxxx"
rc = FREAD(f_id1);
rc = FGET(f_id1, s1);

rc = FPUT(f_id2, s1);
rc = FWRITE(f_id2);
```

and the file opened and identified by the identifier `f_id1` contains the records:

Title	Type	Author	Read?
Origins and Growth of Sociology, The	Soc	Abraham, J H	n y
First Light	N	Ackroyd, Peter	n y
Book of Visions, The	Ref	Albery, Nicholas (Ed)	c n
How to Build a Mind	Sci	Aleksander, Igor	n y
English Common Reader, The	History	Altick, Richard D	a n

then, if the other statements in the `DATA` step obtained each record from the input file, the records written to the file specified in the `FPUT` statement would be:

```
Origin
First
Book o
How to
Defenc
Main C
Main C
```

That is, only the number of characters that match the length of the string `xxxxxxxx` are read from the input file.

If you create a string for *value* using another function, the record in the FDB can be read up to the first separator, or, if you specify a length, up to that length:

```
rname = QUOTE("r");  
rc = FREAD(f_id1);  
rc = FGET(f_id1, rname);  
  
rc = FPUT(f_id2, rname);  
rc = FWRITE(f_id2);
```

In this example, the variable `rname` contains the value created using the `QUOTE` function, which in this case is `"r"`. However, because `FGET` does not have a length specified, only the characters up to the first separator character are obtained from the record and written to the FDB.

If the file opened and identified by the identifier `f_id1` contained the records described in the earlier examples, and if the other statements in the `DATA` step obtained each record from the input file, the records written to the file identified by the identifier `f_id2` would be:

```
Title  Type  Author      Read?  
Origins  
First  
Book  
How  
English  
Water,  
Okri,  
Periodic
```

The records have been read until the first separator character is met; by default, this is space or comma. Tab is not a default separator, as can be seen in the first line.

Basic example

In this example, records are read from a file, and then characters from that record are written to the `log`.

```
DATA _NULL_;  
  i = 0;  
  rc = FILENAME("myfile", "C:\temp\books\books_read.txt");  
  f_id1 = FOPEN("myfile");  
  rname = QUOTE("r");  
  IF f_id1 = 1;  
    DO UNTIL (i = 10);  
      rc = FREAD(f_id1);  
      rc = FGET(f_id1, rname);  
      PUT rname=;  
      i = i + 1;  
    END;  
  rc = FCLOSE(f_id1);  
RUN;
```

This produces the following output:

```
Title Type Author    Read?
Origins
First
Book
How
English
Water,
Defence
Battle
Main
```

In this example, the characters up to the first separator are obtained. By default, the separator is the space character. Because the variables in the records are tab-separated, the title line has no spaces, so the function gets the whole record.

Example – reading from and writing to external files using a mask-type value

In this example, a record is obtained from an external text file (that uses a comma as the separator) and put into the FDB to be written into a file of another type (a plain text file). The separators , used in the input file has been specified. The record obtained will be truncated either at the first separator, or at 30 characters as specified by the length of the string xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.

Note:

Any separators in the first column are removed, but they do not cause the record to truncate at that point.

```
DATA _NULL_;

rc = FILENAME('myfile','C:\temp\books\books_read.txt');
f_id1 = FOPEN('myfile');
rc = FILENAME('myfile2','C:\temp\books\books_read_out.txt');
f_id2 = FOPEN('myfile2','o');

rname = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
rc = FSEP(f_id1,',');

IF f_id1 = 1;
  DO UNTIL (rr = -1);
    rr = FREAD(f_id1);
    rc = FGET(f_id1, rname);
    rc = FPUT(f_id2, rname);
    rc = FWRITE(f_id2);
  END;
RUN;
```


This creates a file named `books_read.txt`, in the directory `C:\temp\books`, that contains the first thirty characters of all records, unless there is a separator first. If the first few lines of the input file look like this:

```
Origins and Growth of Sociology, The Soc Abraham, J H n y
First Light N Ackroyd, Peter n y
How to Build a Mind,Sci Aleksander, Igor n y
Water, Leisure and Culture Culture Anderson, Susan C, and Bruce H Tabb n y
Defence of the Realm, The Hist Andrew, Christopher n y
Main Currents in Sociological Thought Vol1 Soc Aron, Raymond n y
Main Currents in Sociological Thought Vol2 Soc Aron, Raymond n y
```

The first few lines of the `books_read.txt` would look like this:

```
Origins and Growth of Sociolog
First Light
Book of Visions
How to Build a Mind
Water
Defence of the Realm
Main Currents in Sociological
Main Currents in Sociological
```

The first, last and penultimate lines have been truncated after 30 characters; all other lines end where there was either a comma or quote mark, as specified by the `FSEP` function.

Reading from an external file and writing to a dataset

In this example, the records are read from one type of external file (a text file) into a dataset named `example` in the `WORK` library. The pointer is positioned at character 56 of the record, and then 55 characters of the record are obtained from memory and written to the `FDB`.

```
DATA example;
  KEEP x;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile", "s");
  rec = QUOTE("r");
  IF f_id1 = 1;
    DO UNTIL (rr = -1);
      rr = FREAD(f_id1);
      rc = fpos(f_id1, 54);
      rc = FGET(f_id1, rec, 55);
      x = rec;
      OUTPUT;
    END;
  RUN;
```

If the first few lines of the contains these records:

Title	Type	Author
Origins and Growth of Sociology, The	Soc	Abraham, J H
First Light	N	Ackroyd, Peter
Book of Visions, The	Ref	Albery, Nicholas (Ed)
How to Build a Mind	Sci	Aleksander, Igor
Defence of the Realm, The	Hist	Andrew, Christopher
Main Currents in Sociological Thought Vol1	Soc	Aron, Raymond

Main Currents in Sociological Thought Vol2	Soc	Aron, Raymond
--	-----	---------------

The first few lines of `books_read.txt` would contain these records:

Type	Author
Soc	Abraham, J H
N	Ackroyd, Peter
Ref	Albery, Nicholas (Ed)
Sci	Aleksander, Igor
Hist	Andrew, Christopher
Soc	Aron, Raymond
Soc	Aron, Raymond

FILEEXIST

Returns a value that indicates whether a specified file exists, based on its physical location.

➤ **FILEEXIST** ➤ (➤ *physical-location* ➤) ➤

Return type: Numeric

1 if the specified exists, 0 (zero) otherwise.

physical-location

Type: Character

The pathname and filename of the file.

Example

In this example, the function returns a value indicating whether the file exists or not. The value is used to generate a message using `IFC` [↗](#) (page 671). The result is written to the log.

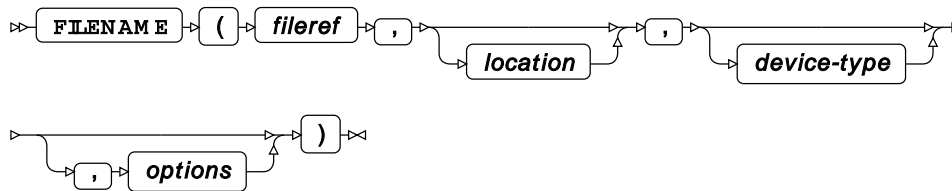
```
DATA _NULL_;
  rc = FILEEXIST("C:\temp\examples.7z");
  yn = IFC(rc, "The file exists", "The file does not exist");
  PUT YN;
  rc = FILEEXIST("C:\temp\notexist.txt");
  yn = IFC(rc, "The file exists", "The file does not exist");
  PUT YN;
RUN;
```

This produces the following output:

```
The file exists
The file does not exist
```

FILENAME

Creates a reference to a file or fileref that can be used in other external file functions, such as `FOPTNAME` and `FINFO`



Return type: Numeric

fileref

Type: Character

The file reference. This is a character string.

location

Optional argument

Type: Character

The pathname and filename of the file. If no value is assigned to this item, the reference

device-type

Optional argument

Type: Character

Specifies the type of device on which the file is stored. The devices can be any of the devices described in *FILENAME statements* in the section *Global statements*, such as DISK, PIPE, HADOOP, and CATALOG.

options

Optional argument

Type: Character

Options associated with the device defined by *device-type*. These options are described in the corresponding entries for the device in the section *FILENAME statements*.

Example

In this example, the function specifies a reference for the specified filename, which is then used by FEXIST to check that the filename exists. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("myfile", "C:\temp\books\books.accdb", disk);  
  rc = FEXIST("myfile");  
  yn = IFC(rc, "The file exists", "The file does not exist");  
  PUT yn;  
  rc = FILENAME("myfile");  
  rc = FEXIST("myfile");  
  yn = IFC(rc, "The file exists", "The file does not exist");  
  PUT yn;  
RUN;
```

This produces the following output:

```
The file exists  
The file does not exist
```

In the first use of the function, the specified file is assigned to the reference `myfile`. In the second use of the function, no file is specified to `myfile`; the file previously assigned is no longer attached to the reference.

FILEREF

Returns a value indicating whether a specified file reference has been assigned.

⇒ **FILEREF** ⇒ (⇒ *fileref* ⇒) ⇒

Return type: Numeric

0	The file reference exists, and the filename associated with it is valid.
2004	The file reference does not exist (that is, the reference specified has not been defined in the DATA step in a FILENAME function).
-2006	The file reference exists (that is, the reference specified has been defined in the DATA step in a FILENAME function), but the file assigned to the reference does not exist.

fileref

Type: Character

A file reference created by [FILENAME](#) (page 1389)

Example

In this example, a file reference for the pathname and filename of a file is created using the `FILENAME` function. The `FILEREF` function is then used to check whether the file reference and the file associated with it exist. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.txt");
  f_id = FOPEN("myfile");
  var = FILEREF("myfile");
  PUT "reference = " var;
  msg = SYSMSG();
  PUT msg;
  var = FILEREF("myfile2");
  PUT "reference = " var;
  msg = SYSMSG();
  PUT msg;
run;
```

This produces the following output:

```
reference = 0
reference = 20004
The fileref myfile2 is not assigned
```

In the first use of the function, the file reference exists and references an existing file, and so returns 0. In the second use of the function, the file reference does not exist, so the value 2004 is returned.

In this example, the `SYSMSG` function has also been used to display the associated messages. No message is associated with the return code 0.

FINFO

Returns the value of a specified property of a file.

➤ **FINFO** ➤ (➤ *file-id* ➤ , ➤ *property* ➤) ➤

Each file has a number of properties or options associated with it, such as the type, or record size, and so on. You can get a list of the properties using the `FOPTNUM` [\(page 1396\)](#) and `FOPTNAME` [\(page 1396\)](#) functions, and then use this function to get the value of a specified property.

The properties available depend on the operating system.

The file is specified using an identifier generated by `FOPEN` [\(page 1393\)](#).

Return type: Character

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [\(page 1393\)](#).

property

Type: Character

The value of a property returned by `FOPTNAME`. The properties returned, and the values associated with them, depend on the operating system and file-type.

Example

In this example, the value of all properties for the specified file are returned. The number of properties is first returned using `FOPTNUM`, the names of the properties are returned using `FOPTNAME`, and the values for each property are then returned using `FINFO`. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.txt");
  z = FEXIST("myfile");
  o = FOPEN("myfile");
  x = FOPTNUM(o);
  DO i = 1 TO x;
    y = FOPTNAME(o,i);
    n = FINFO(o, y);
    PUT "The value of the property " y "is: " n;
  END;
RUN;
```

This produces the following output:

```
The value of the property File Name is: 'C:\temp\books\books_read.txt'
The value of the property Lrecl is: 256
The value of the property Recfm is: V
```

In this example, the file has a record length of 256 bytes and a record format of `V`, that is, variable-length.

FNOTE

Returns an identifier for a record in the File Data Buffer. The record can then be located using `FPOINT` and the identifier.

➤ **FNOTE** ➤ (➤ *file-id* ➤) ➤

This function should only be used with files specified as random access files. No identifier will be returned if the file is specified as a sequential file.

Return type: Numeric

Returns 0 (zero) if there is an error, otherwise the identifier is a positive integer. The integer increases by one for each new note, unless a previous identifier is released by dropping the corresponding note using `DROPNOTE`, in which case `FNOTE` will use the released identifier.

file-id**Type:** Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, all records of a file are read from a specified file into the FDB; the 1000th and 1500th record are noted using `FNOTE`. The result is written to the log. You could subsequently find these records in the FDB using `FPOINT`, if required. See `FPOINT` [↗](#) (page 1398) for an example of using `FNOTE` and `FPOINT` to note a record and then get that record.

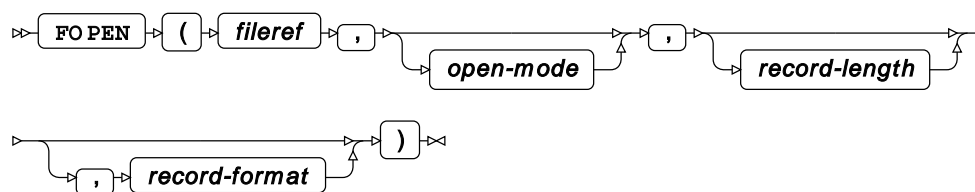
```
DATA _NULL_;
rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
f_id1 = FOPEN("myfile");
rec = QUOTE("r");
z = 1;
IF f_id1 = 1;
DO UNTIL (z = 2010);
rr = FREAD(f_id1);
rc = FGET(f_id1, rec);
IF z = 1000 THEN ni1 = FNOTE(f_id1);
IF z = 1500 THEN ni2 = FNOTE(f_id1);
z = z + 1;
END;
PUT ni1=;
PUT ni2=;
RUN;
```

This produces the following output:

```
ni1=1
ni2=2
```

FOPEN

Returns an identifier for a specified file that can then be used in other file functions to specify the file.



You cannot get an identifier for a file directly. You must first use the `FILENAME` [↗](#) (page 1389) function to specify a reference for a filename, and then use `FOPEN` to specify an identifier for that reference.

Return type: Numeric

fileref

Type: Character

A reference for the filename, specified using the `FILENAME` function.

open-mode

Optional argument

Type: Character

The mode in which the dataset is opened. If the file is to be used as an input file, this can be:

"I"

Random access mode.

"S"

Sequential access mode in which observations are read from beginning to end, but previous observations can be read.

If the file is to be used as an output file, this can be:

"A"

Opens an existing file and appends records to the end of it. If the file does not exist, it is created.

"O"

Opens a file and adds records to it. If the file does not exist, it is created. If the file already exists, the file is overwritten.

.

"U"

Opens a file for update. The file can be read or written to.

Using any other character is not allowed; if you do use a character not in this list, the file is not opened, and the identifier 0 is returned.

record-length

Optional argument

Type: Numeric

The record length defined for the file, in bytes.

record-format

Optional argument

Type: Character

The format of records in the file:

"B"

Binary.

"D"

Default. This is `FB` on MVS, `V` on any other operating system.

"F"

Fixed length.

"P"

Print file.

"V"

Variable length.

Options can be combined where appropriate; for example `FB` for a file with fixed-length binary records.

Using any other character is not allowed; if you do use a character not in this list, the file is not opened, the identifier 0 is returned.

If the file does not exist, the identifier 0 is returned. If the file exists, the identifier 1 is assigned to the first file opened. If another file is then opened, the identifier is set to 2. The identifier increments by one for each file opened in the same `DATA` step unless a previously opened file is closed using `FCLOSE`, in which case the released identifier is used.

All open files are closed and their identifiers are released when the `DATA` step ends.

Example

In this example, the function returns an identifier for the specified folder references. The references are specified using the `FILENAME` function. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\author_srted.wpd");
  f_id=FOPEN("myfile");
  PUT "The identifier is: " f_id;
  rc = FILENAME("myfile2", "C:\temp\books\author_sorted.wpd");
  f_id=FOPEN("myfile2");
  PUT "The identifier is: " f_id;
  rc = FILENAME("myfile3", "C:\temp\Code.sas");
  f_id=FOPEN("myfile3");
  PUT "The identifier is: " f_id;
  y = FCLOSE(f_id);
  rc = FILENAME("myfile4", "C:\temp\examples.7z");
  f_id=FOPEN("myfile4");
  PUT "The identifier is: " f_id;
  rc = FILENAME("myfile5", "C:\temp\short_text_example.pdf");
  f_id=FOPEN("myfile5");
  PUT "The identifier is: " f_id;
RUN;
```

This produces the following output:

```
The identifier is: 0
The identifier is: 1
The identifier is: 2
The identifier is: 2
The identifier is: 3
```

In this example:

- The first identifier returned is 0, because the specified file does not exist.
- The second identifier returned is 1. The specified file exists, so the first identifier is set.
- The third identifier returned is 1, because the specified file exists, and the `FCLOSE` function was used before this `FOPEN` to close the previously opened file and release the identifier.
- The fourth identifier returned is 2. The specified file exists, so the identifier has been incremented by one.
- The fifth identifier returned is 3. The specified file exists, so the identifier has been incremented by one.

FOPTNAME

Returns a specified option or property for an open file.

➤ **FOPTNAME** ➤ (➤ *file-id* ➤ , ➤ *index* ➤) ➤

Each file has a number of properties or options associated with it, such as the type, or record size, and so on. You can find how many properties or options the file has with `FOPTNUM` function, and then use this function to list a specified property or option. The properties or options exist as a list; use this function to select a specific item in the list.

The properties available depend on the operating system.

The file is specified using an identifier generated by `FOPEN` [↗](#) (page 1393).

Return type: Character

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

index

Type: Numeric

The position of the property to return in the list of properties or options.

Example

In this example, the number of properties or options is returned by `FOPTNUM`, and then each property or option is returned using `FOPTNAME`. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("myfile", "C:\temp\books\books_read.txt");  
  f_id=FOPEN("myfile");  
  DO i = 1 TO FOPTNUM(f_id);  
    y = FOPTNAME(f_id,i);  
    PUT "property " i "is " y;  
  END;  
RUN;
```

This produces the following output:

```
property 1 is File Name  
property 2 is Lrecl  
property 3 is Recfm
```

FOPTNUM

Returns the number of properties associated with a specified file.

⇒ **FOPTNUM** ⇒ (⇒ *file-id* ⇒) ⇒

Each file has a number of properties or options associated with it, such as the type, or the originating user, and so on. You can find how many properties or options there are with this function, and then use `DOPTNAME` to list all the properties or options.

The properties available depend on the operating system.

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, the number of properties or options is returned by `FOPTNUM`, and then each property or option is returned using `FOPTNAME`. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("fname", "C:\temp\books\books_read.prn");  
  fid=FOPEN("fname");  
  DO i = 1 TO FOPTNUM(fid);  
    y = FOPTNAME(fid,i);  
    PUT "property " i "is " y;  
  END;  
RUN;
```

This produces the following output:

```
property 1 is File Name  
property 2 is Lrecl  
property 3 is Recfm
```

You can then use `FINFO` to get information about each of the properties, if required. For example, you could use `FINFO` to return the value of `Lrecl`, which contains the maximum record length of the file.

FPOINT

Points the file pointer at a record in the File Data Buffer specified by a note identifier. A note identifier is defined using `FNOTE`.

⇒ **FPOINT** ⇒ (⇒ *file-id* ⇒ , ⇒ *note-id* ⇒) ⇒

Return type: Numeric

Returns 1 if the note identifier exists, 0 (zero) otherwise.

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

note-id

Type: Numeric

A note identifier, created using `FNOTE` [↗](#) (page 1392).

Example

In this example, the specified file is read from beginning to end; the 1000th character is written to the log and noted using `FPOINT`. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile");
  rec = QUOTE("r");
  z = 1;
  IF f_id1 = 1;
    DO UNTIL (z = 2010);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rec);
      IF z = 1000 THEN ni = FNOTE(f_id1);
      IF z = 1000 THEN PUT rec=;
      z = z + 1;
    END;
  rc = FPOINT(f_id1, ni);
  rr = FREAD(f_id1);
  rc = FGET(f_id1, rec);
  PUT rec=;
  rc = DROPNOTE(f_id1, ni);
  rc = FCLOSE(f_id1);
RUN;
```

This produces the following output:

```
rec=Breakfast
rec=Breakfast
```

The 1000th record is written to the log during the `DO` loop. The record is also noted with `FNOTE`. `FPOINT` is subsequently used to reposition the file pointer at that record, at which point it is read from the FDB using `FREAD` and `FGET`.

FPOS

Positions the file pointer at the specified column in the current row in the file.

```
⇒ FPOS ( file-id , position ) ⇒
```

For an input file, the current record will be read from the specified position. For an output file, the current record will be written at the specified position.

Return type: Numeric

Returns 0 if the record exists, 1 if not.

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

position

Type: Numeric

The position in the record at which to start reading or writing.

The position of this function in the `DATA` step is important. It must come before an `FPUT` (which itself must come before an `FGET`).

Example – positioning the pointer for input

In this example, a string is saved to a variable, and then that variable is written to a file at the specified position in the record.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\example.txt");
  f_id1 = FOPEN("myfile", "o");
  wrrec = "A record for the file";
  rc = FPOS(f_id1, 10);
  rc = FPUT(f_id1, wrrec);
  rc = FWRITE(f_id1);
  rc = FCLOSE(f_id1);
RUN;
```

This creates a file named `example.txt` in the directory `C:\temp`. The file contains the text of the specified variable, starting at column ten of the record:

```
A record for the file
```

Example – positioning the pointer for input

In this example, the record in the file created in the previous example is read into the FDB, and then ten characters from that record are obtained from the FDB starting at character position 12. The result is written to another file.

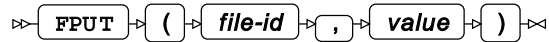
```
DATA _NULL_;
  rec = QUOTE("r");
  rc = FILENAME("ifile", "C:\temp\example.txt");
  inf = FOPEN("ifile");
  rc = FILENAME("ofile", "C:\temp\example_o.txt");
  outf = FOPEN("ofile", "o");
  rc = FREAD(inf);
  rc = FPOS(inf, 12);
  rc = FGET(inf, rec, 10);
  rc = FPUT(outf, rec);
  rc = FWRITE(outf);
  rc = FCLOSE(inf);
  rc = FCLOSE(outf);
RUN;
```

This creates a file named `example_o.txt` in the directory `C:\temp`. The file contains one record, consisting of the specified characters from the input file:

```
record for
```

FPUT

Puts (writes) a record into the File Data Buffer (FDB).



Return type: Numeric

This function would typically be used with the `FWRITE` function that would subsequently write the record put into the FDB to a specified file.

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [\(page 1393\)](#).

value

Type: Character

A variable containing the data to be put into the FDB.

The data put to the FDB will be affected by whether the argument supplied is a string or a variable. See the section below for details.

The way in which *value* is specified in the corresponding `FGET` will affect the data written to the variable *value*; see `FGET` [\(page 1383\)](#) for details.

Basic example

In this example, a string is saved to a variable, then that variable is written to a file using `FPUT` and `FWRITE`.

```
DATA _NULL_;

rcfn = FILENAME('myfile', 'C:\temp\example.txt');
f_id1 = FOPEN('myfile', 'o');

wrrec = 'A record for the file';

rcfp = FPUT(f_id1, wrrec);
rcfw = FWRITE(f_id1);

RUN;
```

This creates a file named `example.txt` in the directory `C:\temp` that contains the text:

```
A record for the file
```

Example – reading from and writing to external files

In this example, selected records are read from one type of external file (.prn) into a file of another type (a .txt file). Those records containing a specified string are written to a new file.

```
DATA _NULL_;
  rc = FILENAME('myfile','C:\temp\books\books_read.prn');
  f_id1 = FOPEN('myfile');
  rc = FILENAME('myfile2','C:\temp\books\sociology_books.txt');
  f_id2 = FOPEN('myfile2','o');
  rname = QUOTE('r');
  IF f_id1 = 1;
    DO UNTIL (rr = -1);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rname, 110);
      IF (contains(rname, ' Soc ')) THEN
        DO;
          rc = FPUT(f_id2, trim(rname));
          rc = FWRITE(f_id2);
        END;
      END;
    END;
  RUN;
```

This creates a file named `sociology_books.txt` in the directory `C:\temp\books` that contains a list of all the records containing the string `Soc`, which indicates a book in the genre of sociology. The first four lines of the file would look like this:

Origins and Growth of Sociology, The	Soc	Abraham, J H
Main Currents in Sociological Thought Vol1	Soc	Aron, Raymond
Main Currents in Sociological Thought Vol2	Soc	Aron, Raymond
Sociology of Science	Soc of Sci	Barnes, Barry (Ed)

The example also shows a variable created using the `QUOTE` function, and how the length of the record required is specified using the `FGET` function.

Example – reading from and writing to external files using a mask-type value

In this example, a record is obtained from an external text file (that uses a comma as the separator) and put into the FDB to be written into a file of another type (a plain text file). The separators , used in the input file has been specified. The record obtained will be truncated either at the first separator, or at 30 characters as specified by the length of the string `xx`.

Note:

Any separators in the first column are removed, but they do not cause the record to truncate at that point.

```
DATA _NULL_;

rc = FILENAME('myfile', 'C:\temp\books\books_read.txt');
f_id1 = FOPEN('myfile');
rc = FILENAME('myfile2', 'C:\temp\books\books_read_out.txt');
f_id2 = FOPEN('myfile2', 'o');

rname = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
rc = FSEP(f_id1, ',');

IF f_id1 = 1;
DO UNTIL (rr = -1);
rr = FREAD(f_id1);
rc = FGET(f_id1, rname);
rc = FPUT(f_id2, rname);
rc = FWRITE(f_id2);
END;

RUN;
```

This creates a file named `books_read.txt`, in the directory `C:\temp\books`, that contains the first thirty characters of all records, unless there is a separator first. If the first few lines of the input file look like this:

```
Origins and Growth of Sociology, The Soc Abraham, J H n y
First Light N Ackroyd, Peter n y
How to Build a Mind, Sci Aleksander, Igor n y
Water, Leisure and Culture Culture Anderson, Susan C, and Bruce H Tabb n y
Defence of the Realm, The Hist Andrew, Christopher n y
Main Currents in Sociological Thought Vol1 Soc Aron, Raymond n y
Main Currents in Sociological Thought Vol2 Soc Aron, Raymond n y
```

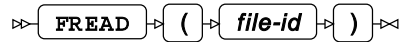
The first few lines of the `books_read.txt` would look like this:

```
Origins and Growth of Sociolog
First Light
Book of Visions
How to Build a Mind
Water
Defence of the Realm
Main Currents in Sociological
Main Currents in Sociological
```

The first, last and penultimate lines have been truncated after 30 characters; all other lines end where there was either a comma or quote mark, as specified by the `FSEP` function.

FREAD

Reads a record from a specified file to the File Data Buffer (FDB).



The file to be read is specified using the file identifier created by the `FOPEN` function. This function is typically used with an `FGET` function, which gets the record from the FDB and puts it into a variable.

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, the first ten records are read from a specified file into the FDB; the first 100 characters of the records are then obtained from the FDB using `FGET`. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile");
  i = 0;
  rec = QUOTE("r");
  IF f_id1 = 1;
    DO UNTIL (i eq 10);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rec, 100);
      PUT rec;
      i = i + 1;
    END;
  RUN;
```

This produces the following output:

Title	Type	Author
Origins and Growth of Sociology, The	Soc	Abraham, J H
First Light	N	Ackroyd, Peter
Book of Visions, The	Ref	Albery, Nicholas (Ed)
How to Build a Mind	Sci	Aleksander, Igor
Defence of the Realm, The	Hist	Andrew, Christopher
Main Currents in Sociological Thought Vol1	Soc	Aron, Raymond
Main Currents in Sociological Thought Vol2	Soc	Aron, Raymond

FRECCNT

Returns the number of records in a file opened with FOPEN.

➤ **FRECCNT** ➤ (➤ *file-id* ➤) ➤

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using FOPEN [↗](#) (page 1393).

If you want to find the number of records before performing some other operation on the file, such as reading records from the file sequentially using a DO loop, you will have to rewind the file to the first record using FREWIND.

Example

In this example, the number of records in the specified file is returned. The result is written to the log.

```
DATA _NULL_;  
  rc = FILENAME("mydir","C:\temp");  
  dir_id=DOPEN("mydir");  
  DO i = 1 TO DOPTNUM(dir_id);  
    y = DOPTNAME(dir_id,i);  
    PUT "property " i "is " y;  
  END;  
RUN;
```

This produces the following output:

```
The number of records in the file is: 2011
```

FREWIND

Positions the file pointer before the first record.

➤ **FREWIND** ➤ (➤ *file-id* ➤) ➤

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

You can rewind input or output files. In some file modes, if you rewind an output file, existing records are deleted and records are added starting at the first record; in others, records will not be deleted, and records will be appended at the end of the file.

Basic example

In this example, the first and last records of an external file are read up to the first separator, and then written to the log file. The file is then rewound to the first record, and the first record is again read up to the first separator, and then written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile");
  rec = QUOTE("r");
  z = 1;
  IF f_id1 = 1;
    DO UNTIL (z = 2010);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rec);
      IF z = 1 THEN PUT "The first record is: " rec;
      IF z = 2008 THEN PUT "The last record is: " rec;
      z = z + 1;
    END;
  rc = FREWIND(f_id1);
  rr = FREAD(f_id1);
  rc = FGET(f_id1, rec, 50);
  PUT "After rewind the record read is: " rec;
  rc = FCLOSE(f_id1);
RUN;
```

This produces the following output:

```
Characters from the first record: Title
Characters from the last record: Middle
After rewind, characters are: Title
```

Example with output

In this example, a file is read from an input file until the 10th record, and the first fifty characters of each record are written to another external file. The file is then rewound, and the same operation is performed again.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  f_id1 = FOPEN("myfile");
  rc = FILENAME("myfile2", "C:\temp\books\books_read_out.txt");
  f_id2 = FOPEN("myfile2", "a");
  rec = QUOTE("r");
  z = 0;
  IF f_id1 = 1;
    DO UNTIL (z=10);
      rr = FREAD(f_id1);
      rc = FGET(f_id1, rec, 50);
      rc= FPUT(f_id2, rec);
    END;
  rc = FREWIND(f_id1);
  rr = FREAD(f_id1);
  rc = FGET(f_id1, rec, 50);
  rc= FPUT(f_id2, rec);
END;
```

```
rc = FWRITE(f_id2);
z = z + 1;
END;
z= 0;
rc = FREWIND(f_id1);
DO UNTIL (z=10);
    rr = FREAD(f_id1);
    rc = FGET(f_id1, rec,50);
    rc = FPUT(f_id2, rec);
    rc = FWRITE(f_id2);
    z = z + 1;
END;
rc = FCLOSE(f_id1);
rc = FCLOSE(f_id2);
RUN;
```

The following records will be written to the file:

```
Title
Origins and Growth of Sociology, The
First Light
How to Build a Mind
English Common Reader, The
Water, Leisure and Culture
Defence of the Realm, The
Main Currents in Sociological Thought Vol1
Main Currents in Sociological Thought Vol2
Nightfall
Title
Origins and Growth of Sociology, The
First Light
How to Build a Mind
English Common Reader, The
Water, Leisure and Culture
Defence of the Realm, The
Main Currents in Sociological Thought Vol1
Main Currents in Sociological Thought Vol2
Nightfall
```

Because `FREWIND` has been specified the file pointer has been repositioned at the first record in the file, and then ten records read from that point in the following `DO` loop.

If `FREWIND` had not been specified, the output would instead have been:

```
Title
Origins and Growth of Sociology, The
First Light
How to Build a Mind
English Common Reader, The
Water, Leisure and Culture
Defence of the Realm, The
Main Currents in Sociological Thought Vol1
Main Currents in Sociological Thought Vol2
Nightfall
Periodic Kingdom, The
Behind the Scenes at the Museum
Case Histories
Emotionally Weird
Not the End of the World
One Good Turn
```

```
Started Early, Took my Dog  
When Will there be Good News?  
Knowledge and Explanation in History  
Emma
```

That is, the input file would not have been rewound, and the first `FREAD` in the second `DO` loop would read the record after the last record read in the previous `DO` loop.

FRLEN

Returns the length of the current record.

➤ **FRLEN** ➤ (➤ *file-id* ➤) ➤

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

Example

In this example, the length of each record in a file is checked, and the value of the longest is stored and then written to the log.

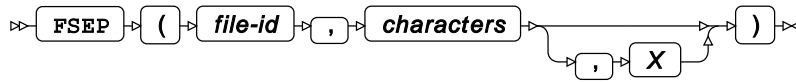
```
DATA _NULL_;  
  
rc = FILENAME("myfile", "C:\temp\books\books_read.prn");  
f_id1 = FOPEN("myfile");  
  
IF f_id1 = 1;  
DO UNTIL (gno = -1);  
  gno = FREAD(f_id1);  
  r1 = FRLEN(f_id1);  
  IF r1 > tr1 THEN tr1 = r1;  
END;  
  
PUT "The longest record contains: " tr1 "characters";  
RUN;
```

This produces the following output:

```
The longest record contains: 179 characters
```

FSEP

Specifies the separator for the record obtained by an `FREAD` function.



Return type: Numeric

Returns 1 if the function was successful, 0 (zero) otherwise.

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

characters

Type: Character

One or more characters that specify the separator. For example, you might specify comma `(,)` or quotation mark `(")`, or you might specify both `(, ")`.

X

Optional argument

Must be `"X"`.

"X"

Specifies that the separator in the argument *characters* is a hexadecimal value. For example, if the separator is the tab character, you can specify this instead with the string `"09"`

Example

In this example, the separator is defined as the tab character. The result is written to the log.

```
DATA _NULL_;

rc = FILENAME("myfile", "C:\temp\books\books_read.txt");
f_id1 = FOPEN("myfile");
i = 0;

rec = QUOTE("r");

IF f_id1 = 1;
DO UNTIL (i eq 12);
  rr = FREAD(f_id1);
  rc = FSEP(f_id1, "09", "X");
  rc = FGET(f_id1, rec);
  PUT rec;
  i = i + 1;
END;

RUN;
```

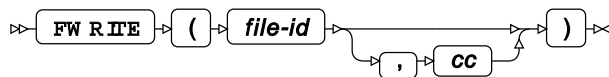
This produces the following output:

```
Title
Origins and Growth of Sociology, The
First Light
How to Build a Mind
English Common Reader, The
Defence of the Realm, The
Main Currents in Sociological Thought Vol1
Main Currents in Sociological Thought Vol2
Nightfall
Okri, Ben
Periodic Kingdom, The
Behind the Scenes at the Museum
```

Each record is read up to the first tab character.

FWRITE

Writes a record from the File Data Buffer to a specified file.



The file to which the record is written is specified using the file identifier created by the `FOPEN` function. This function is typically used with an `FPUT` function, which puts the record into the File Data Buffer.

Return type: Numeric

file-id

Type: Numeric

The identifier of a file previously opened using `FOPEN` [↗](#) (page 1393).

cc

Optional argument

Type: Character

A character used to specify the type of carriage control to be inserted at the end or beginning of the written record. This can be:

- 0 Inserts two carriage return and line feed combinations (double spacing)
- Inserts three carriage return and line feed combinations (triple spacing)
- 1 Inserts a form-feed at the beginning of the line. This has an effect in some file formats, such as Microsoft Word, where a page break is inserted. In other file formats, a character representing the control character might be visible. See examples below.
- + If the output is directed to a printer, or is a printer file, or to some displays, the record will overwrite the previous record. This can be used to create boldface or underlined text. This has no effect in most files.

By default, the record is written to the next line (carriage return and line feed).

Basic example

In this example, a string is saved to a variable, then that variable is written to a file using `FPUT` and `FWRITE`.

```
DATA _NULL_;

rcfn = FILENAME('myfile', 'C:\temp\example.txt');
f_id1 = FOPEN('myfile', 'o');

wrrec = 'A record for the file';

rcfp = FPUT(f_id1, wrrec);
rcfw = FWRITE(f_id1);

RUN;
```

This creates a file named `example.txt` in the directory `C:\temp` that contains the text:

```
A record for the file
```

Writing a file with carriage control characters

In this example, records are read from an external file, and then the first ten records are written to another external file. Various carriage control characters are also appended to the written records.

```
DATA _NULL_;
```

```
rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
f_id1 = FOPEN("myfile", "s");

rc = FILENAME("myfile2", "C:\temp\books\books_read_out.txt");
f_id2 = FOPEN("myfile2", "o");

rec = quote("r");
z = 0;

IF f_id1 = 1;
  DO UNTIL (z=10);

    rr = FREAD(f_id1);
    rc = FGET(f_id1, rec, 50);

    rc= FPUT(f_id2, rec);
    if z <= 2 then rc = FWRITE(f_id2);
    if z GT 2 & z LE 5 then rc = FWRITE(f_id2, "-");
    if z GT 5 & z LE 7 then rc = FWRITE(f_id2, "0");
    if z GT 7 & z LE 10 then rc = FWRITE(f_id2, "1");

    z = z + 1;

  END;

rc = fclose(f_id1);
rc = fclose(f_id2);

RUN;
```

This creates the following text in the file `books_read_out.txt`:

```
Title
Origins and Growth of Sociology, The
First Light

How to Build a Mind

English Common Reader, The

Water, Leisure and Culture

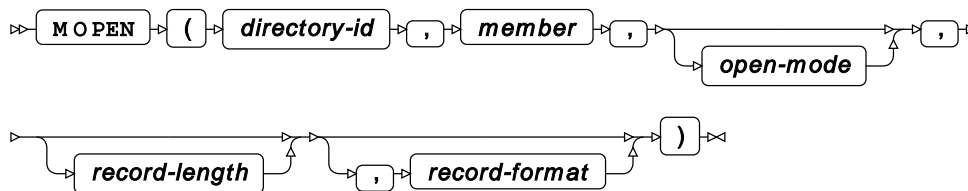
Defence of the Realm, The

Main Currents in Sociological Thought Vol1
↑Main Currents in Sociological Thought Vol2
↑Nightfall
```

The first two lines have default spacing. The next three lines are followed by three line feeds. The next two lines are followed by two line feeds. The final two lines are prepended with a form feed. This has no affect in the text file, but the ↑ character shows where the form feeds are. If you were to open this file in Microsoft Word, the last two records would be on new pages (that is, they would be preceded by a Word page break).

MOPEN

Returns an identifier for a specified member or file that can then be used in other file functions to specify the member or file.



Opens a member or file in a directory or folder, and returns an identifier. The containing file or folder must first have been opened using `DOPEN` [\(page 1374\)](#).

Return type: Numeric

directory-id

Type: Numeric

Identifier of the directory or folder containing the file to be opened. The identifier is created using `DOPEN` [\(page 1374\)](#).

member

Type: Character

open-mode

Optional argument

Type: Character

The mode in which the dataset is opened. If the file is to be used as an input file, this can be:

"I"

Random access mode.

"S"

Sequential access mode in which observations are read from beginning to end, but previous observations can be read.

If the file is to be used as an output file, this can be:

"A"

Opens an existing file and appends records to the end of it. If the file does not exist, it is created.

"O"

Opens a file and adds records to it. If the file does not exist, it is created. If the file already exists, the file is overwritten.

"U"

Opens a file for update. The file can be read or written to.

Using any other character is not allowed; if you do use a character not in this list, the file is not opened, and the identifier 0 is returned.

record-length

Optional argument

Type: Numeric

record-format

Optional argument

Type: Character

The format of records in the file:

"B"

Binary.

"D"

Default. This is `FB` on MVS, `V` on any other operating system.

"F"

Fixed length.

"P"

Print file.

"V"

Variable length.

If the file does not exist, the identifier 0 is returned. If the file exists, the identifier 1 is assigned to the first file opened. If another file is then opened, the identifier is set to 2. The identifier increments by one for each file opened in the same `DATA` step unless a previously opened file is closed using `FCLOSE`, in which case the released identifier is used.

All open files are closed and their identifiers are released when the `DATA` step ends.

Example

In this example, the function returns an identifier for the specified folder references. The references are specified using the `FILENAME` function. The result is written to the log.

```
DATA _NULL_;  
  d_id = DOPEN("mydir");  
  f_id=MOPEN(d_id,"books_read.txt");  
  PUT "The identifier is: " f_id;  
  f_id=MOPEN(d_id,"books_read.prn");  
  PUT "The identifier is: " f_id;  
  rc = fclose(f_id);  
  f_id=MOPEN(d_id,"books_read.csv");  
  PUT "The identifier is: " f_id;  
RUN;
```

This produces the following output:

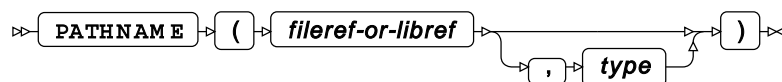
```
The identifier is: 0
```

In this example:

- The first identifier returned is 1. The specified file exists, so the first identifier is set.
- The second identifier returned is 2.
- The third identifier returned is 2 because the specified file exists, and the `FCLOSE` function was used before this `MOPEN` to close the previously opened file and release the identifier.

PATHNAME

Returns the full pathname for a folder or filename that has already been specified using `FILENAME` or `LIBNAME`.



Whether the function returns a pathname to a folder or directory, or to a pathname including a filename, depends on whether you specify:

- A reference to a library, created with `LIBNAME` [↗](#) (page 618)
- A reference to a file or folder, created with `FILENAME` [↗](#) (page 1389)

and the option you also specify.

Return type: Character

fileref-or-libref

Type: Character

The file reference specified for a pathname or filename in a `FILENAME` function , or the libref specified using `LIBNAME`.

type

Optional argument

Type: Character

The type of reference. If the reference does not match the type of reference, an error message is returned.

"L"

fileref-or-libref is a libref. This is the default if *fileref-or-libref* is a libref.

"F"

fileref-or-libref is a file reference. This is the default if *fileref-or-libref* is a file reference.

If `F` is specified, the function assumes that the value to be returned is either:

- A filename, including the pathname, if you specified a reference created with `FILENAME` that refers to a file
- A folder or directory name, including the pathname, if you specified a reference created with `FILENAME` that refers to a folder or directory

If you specify `F` for a libref, an error is returned.

If you specify `L` for *type*, the function assumes that the value to be returned is a pathname previously defined using `LIBNAME`. If you specify `L` for a file identifier, an error is returned.

Basic example

In this example, the pathname and filename of a file are specified using the `FILENAME` function. The `PATHNAME` function is then used to return those names.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\books_read.prn");
  rc = FILENAME("mydir", "C:\temp\books");
  pn = PATHNAME("mydir");
  PUT "The pathname is: " pn;
  pn = PATHNAME("myfile");
  PUT "The pathname is: " pn;
RUN;
```

This produces the following output:

```
The pathname is: C:\temp\books
The pathname is: C:\temp\books\books_read.prn
```

Example – using options and creating errors

In this example, the pathname and filename of a file are specified using the `FILENAME` function, and a libref is created using `LIBNAME`. The `PATHNAME` function is then used to return the pathnames, using *type* to specify whether the pathname to be returned was set using `FILENAME` or `LIBNAME`.

```
libname books 'C:\temp\books';
DATA _NULL_;

rc = FILENAME("myfile", "C:\temp\books\books_read.prn");

pn = PATHNAME("myfile");
PUT "The pathname is a FILENAME ref - using default option: " pn;

pn = PATHNAME("myfile", "F");
PUT "The pathname is a FILENAME ref - using F option: " pn;

pn = PATHNAME("myfile", "L");
PUT "The pathname is a FILENAME ref - using L option:" pn;

PUT "=====";

pn = PATHNAME("books");
PUT "The pathname is a libref - using default option: " pn;

pn = PATHNAME("books", "F");
PUT "The pathname is a libref - using F option: " pn;

RUN;
```

This produces the following output:

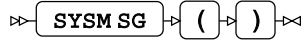
```
The pathname is a FILENAME ref - using default option: C:\temp\books\books_read.prn
The pathname is a FILENAME ref - using F option: C:\temp\books\books_read.prn
NOTE: Argument 1 to function PATHNAME at line 4567 column 8 is invalid
The pathname is a FILENAME ref - using L option:
=====
The pathname is a libref - using default option: C:\temp\books
NOTE: Argument 1 to function PATHNAME at line 4575 column 8 is invalid
The pathname is a libref - using F option:
```

The example shows where that if *type* is the default or `F`, the pathname is returned successfully if used with a reference created using `FILENAME`, but that if used with `L`, no pathname is returned and an error message is generated.

Similarly, if *type* is the default (equivalent to `L`), the pathname is returned successfully if used with a reference created using `LIBNAME`, but if used with `F`, no pathname is returned and an error message is generated.

SYSMSG

Returns an error message for the external file functions.



This function has no arguments. It returns a message based on the numeric return code generated by external file functions. If the external file function is successful, the code generated is 0, and no message is returned.

Return type: Character

Example

In this example, a file reference for the pathname and filename of a file is created using the `FILENAME` function. The `FILEREF` function is then used to check whether the file reference and the file associated with it exist. The result is written to the log.

```
DATA _NULL_;
  rc = FILENAME("myfile", "C:\temp\books\book_read.txt");
  f_id = FOPEN("myfile");
  var = FILEREF("myfile");
  PUT "reference = " var;
  msg = SYSMSG();
  PUT msg;
  var = FILEREF("myfile2");
  PUT "reference = " var;
  msg = SYSMSG();
  PUT msg;
RUN;
```

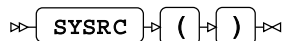
This produces the following output:

```
reference = -20006
The physical file C:\temp\books\book_read.txt associated with fileref myfile does
not exist
reference = 0
```

In the first use of the function, the file reference exists, but the file does not, so the value `-2006` is returned. In the second use of the function, the file reference exists and references an existing file, so `0` is returned.

SYSRC

Returns the error code for an external file function.



This function has no arguments. It returns the numeric code returned by the external file function last executed. If the external file function is successful, the code generated is 0.

Return type: Numeric

Example

In this example, a file reference for the pathname and filename of a file is created using the `FILENAME` function. The `FILEREF` function is then used to check whether the file reference and the file associated with it exist. The result is written to the log.

```
DATA _NULL_;

rc = FILENAME("myfile", "C:\temp\books\books_read.txt");
f_id = FOPEN("myfile");

var = FILEREF("myfile");
rc = SYSRC();
PUT "return code is = " rc;
msg = SYMSG();
PUT msg;

rc = FILENAME("myfile", "C:\temp\books\book_read.txt");
f_id = FOPEN("myfile");

var = FILEREF("myfile");
rc = SYSRC();
PUT "return code is = " rc;
msg = SYMSG();
PUT msg;

var = FILEREF("myfile2");
rc = SYSRC();
PUT "return code is = " rc;
msg = SYMSG();
PUT msg;

RUN;
```

This produces the following output:

```
reference = 0

reference = -20006
The physical file C:\temp\books\book_read.txt associated with FILEREF myfile does
not exist
reference = 20004
The FILEREF myfile2 is not assigned
```

In the first use of the function, the file reference exists and references an existing file, and so returns 0. In the second use of the function, the file reference exists, but the file does not, so the value -2006 is returned. In the third use of the function, the file reference does not exist, so the value 2004 is returned. `SYSRC` is used to return a code, and `SYMSG` is used to display the messages associated with those return codes.

External module functions and CALL routines

Execute code stored in external modules.

The external modules can be any compiled program in a file or library, such as Windows functions, user-created C programs and routines, compiled COBOL programs, and so on. A module can be in a dynamic link library (.dll) on Windows, or a shared object (.so) file on Linux. On z/OS, the module can be a fetchable module.

To call a module on any platform except z/OS, any options or parameters required by the module, and the type of data output by the module must be defined an attribute table. The attribute table is stored as text in a file or in a catalog entry referenced by the name SASCBTBL. The format of the attribute table is described below. The file referenced by SASCBTBL can contain more than one attribute table.

On z/OS, you do not have to define SASCBTBL, as all arguments are passed by address, and each program comprises a module.

Note:
 If the modules are contained in a Windows .dll, the .dll must contain an exports table that lists the modules that can be accessed.

The attribute table ↗	1420
MODULEN ↗	1425
Runs an external module that returns a numeric value.	
MODULEC ↗	1428
Runs an external module that returns a character value.	
CALL MODULE ↗	1431
Runs an external module that does not return a value.	

The attribute table

To enable an external module to be called from a WPS program, on all platforms except z/OS, you must create an attribute table that defines the parameters and outputs of the module. This table is then used by the external routine functions to define the values specified in them.

For example, it is possible to call the Windows CopyFileA function using MODULEN. The attribute table would define how many parameters are required by CopyFileA, the format of those parameters, what the output will be, and so on. This enables the MODULEN function to correctly pass parameters to the function and receive output, such as a return value, from the function.

The attribute table must be stored in a text file or catalog entry that is referenced with the name `SASCBTBL`. This can contain as many attribute tables as required. An attribute table has the following format:

```
ROUTINE name-of-module
MODULE = libname
MINARG = min-args
MAXARG = max-args
CALLSEQ = val-or-addr
STACKORDER = arg-order
STACKPOP = memorymanager
RETURNS = return-type
;
ARG n action arg-type required addressing FDSTART FORMAT=format;
...
```

where:

<i>name-of-module</i>	The name of the module. For example, if this attribute table provides the information required for a Windows function, the name of the module might be <code>CopyA</code> or <code>DeleteA</code> . The name specified here is the name will be used in <code>MODULEN</code> , <code>MODULEC</code> or <code>CALL MODULE</code> .
<i>libname</i>	The name of the library that contains the module. This might be a <code>.dll</code> or <code>.so</code> file. For example, if the module you are calling is a Windows function, the library might be <code>KERNEL32</code> . The library should be in a folder specified on the <code>PATH</code> environment variable, or in the same folder as the program that contains the external function.
<i>min-args</i>	The minimum number of arguments required by the module. The module might not require all arguments; that is, some might be optional. Arguments can be defined as optional by setting the <code>NOTREQD</code> option for <code>ARG</code> .
<i>max-args</i>	The maximum number of arguments required by the module.
<i>val-or-addr</i>	The method used to pass values to the module. Can be: BYVALUE Values are passed directly. BYADDR A reference to the memory address that contains the value is passed. This is the default. The method specified here applies to all arguments. You can, however, override this setting for individual arguments by setting the <i>addressing</i> option of <code>ARG</code> .
<i>arg-order</i>	Specifies the order in which the arguments specified for the module are read onto the stack. This can be:

R2L

The arguments defined for the function are placed on the stack to be read by the module in right to left order. The last argument is therefore first on the stack, and the first argument last.

L2R

The arguments defined for the function are placed on the stack to be read by the module in left to right order. The first argument is therefore first on the stack, and the last argument last. This is the default.

Note:

This option is currently unsupported.

memorymanager

Specifies whether the called module, or the calling WPS program, will update the stack pointer on return from the module. The value can be `CALLER` (the calling program) or `CALLED` (the called module). The default is `CALLER`.

return-type

The type of the value that will be returned by the module. This can be:

CHAR*n*

Pointer to a character string of *n* bytes. *n* is mandatory.

DOUBLE

Double-precision floating-point number. Maximum 8 bytes.

DBLPTR

Pointer to a double-precision floating-point number. Maximum 8 bytes.

INT32

Signed integer. Maximum 4 bytes.

UINT32

Unsigned integer. Maximum 4 bytes.

INT64

Signed integer. Maximum 8 bytes.

UINT64

Unsigned integer. Maximum 8 bytes.

LONG

Long integer. Maximum 4 bytes.

ULONG

Unsigned long integer. Maximum 4 bytes.

SHORT

Short integer. Maximum 2 bytes.

	<p>USHORT</p> <p>Unsigned short integer. Maximum 2 bytes.</p>
<i>n</i>	<p>For each argument that is required by the module, an <code>ARG</code> parameter is defined. The arguments are numbered, starting from 1; the number defines the position of the argument in the command line.</p>
<i>action</i>	<p>Can be:</p> <p>INPUT</p> <p>The value in the argument is passed to the module.</p> <p>OUTPUT</p> <p>The argument receives a value passed back from the module.</p> <p>UPDATE</p> <p>The argument can contain a value that is passed to a module, and then a value returned from the module.</p> <p><code>OUTPUT</code> and <code>UPDATE</code> can only be specified if their values are variables; they cannot be used if the values are constants or expressions.</p>
<i>arg-type</i>	<p>The type of the argument:</p> <p>CHAR</p> <p>The argument contains character data.</p> <p>NUM</p> <p>The argument contains numeric data.</p>
<i>required</i>	<p>Optional. Specifies whether the argument is optional:</p> <p>REQUIRED</p> <p>The argument is required by the module. This is the default.</p> <p>NOTREQD</p> <p>The argument is not required by the module.</p>
<i>addressing</i>	<p>Optional. Specifies how the value of the argument is passed to the module:</p> <p>BYVALUE</p> <p>The value is passed directly.</p> <p>BYADDR</p> <p>A reference to the memory address that contains the value to be passed. This is the default.</p>

FDSTART Optional. The value of the argument is a pointer to a block of values grouped into a structure. All subsequent arguments are assumed to contain values that are successive elements of the structure until another **ARG** with the option **FDSTART** is encountered.

format The SAS language storage format of the value; for example `$CSTR200.` or `RB8.` If the argument is receiving data from the module (that is, it has been specified as **OUTPUT** or **UPDATE**), then this option should be a valid informat.

The format or informat can be any SAS language format or informat, or a user-defined format.

For example, if you want to use the Windows `CopyFileA` function from a **DATA** step, you would use it through the **MODULEN** function, because `CopyFileA` returns a numeric value indicating success or failure. First, however, you would need to define the following attribute table in the file `SASCBTBL`:

```
ROUTINE CopyFileA
  MODULE = KERNEL32
  MINARG = 3
  MAXARG = 3
  STACKPOP = CALLER
  RETURNS = USHORT
  ;
  ARG 1 INPUT CHAR FORMAT=$CSTR200.;
  ARG 2 INPUT CHAR FORMAT=$CSTR200.;
  ARG 3 INPUT NUM FORMAT=PIB4. BYVALUE;
```

This specifies that the function you want to use:

- Is named `CopyFileA` (which is the ANSI version of `CopyFile`)
- Is in the `KERNEL32` Windows module
- Must have three arguments
- Returns an unsigned short

The stack pointer will be updated by the calling **DATA** step.

The first two arguments (the file to copy and the destination filename) are character values; the third argument is a number that specifies what should happen if the file already exists.

You could then specify **MODULEN** in a **DATA** step to use the `CopyFileA` function:

```
DATA _NULL_;
  rc = MODULEN('*E', 'CopyFileA', 'c:\temp\logfile', 'c:\temp\logfile2', 0);
RUN;
```

Note:

You can get information on the parameters required by Windows functions from the Microsoft developer Web site.

Creating and referencing SASCBTBL

The file referenced by SASCBTBL is a plain text file. This file can be created and stored on your computer or on a network, and referenced using a filename reference. For example, you could create an attributes table on Windows in a filename attributes in c:\temp, and then use a filename reference to access it:

```
FILENAME SASCBTBL "c:\temp\attributes";
DATA _NULL_
    rc = MODULEN("*E", "CopyFileA", "c:\temp\test.txt", "c:\temp\test2.txt",0);
RUN;
```

You can also create an attributes file using a DATA step. You could, for example, write the attribute file required by a module to a catalog. For example:

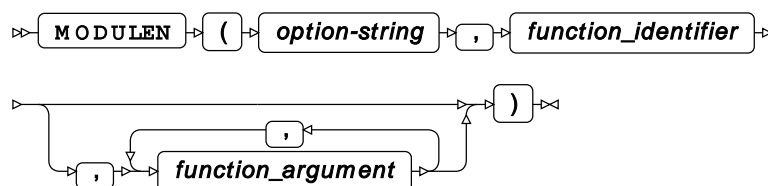
```
FILENAME SASCBTBL CATALOG "WORK.ATTRIBUTES.COPYFILE.SOURCE"
DATA _NULL_;
    FILE SASCBTBL;
    PUT 'routine CopyFileA';
    PUT 'module = KERNEL32';
    PUT 'minarg = 3';
    PUT 'maxarg = 3';
    PUT 'stackpop = called';
    PUT 'returns = ushort';
    PUT ';';
    PUT 'arg 1 input char format=$cstr200.';
    PUT 'arg 2 input char format=$cstr200.';
    PUT 'arg 3 input num format pib4. byvalue';
run;
DATA _NULL_
    rc = MODULEN("*E", "CopyFileA", "c:\temp\test.txt", "c:\temp\test2.txt",0);
RUN;
```

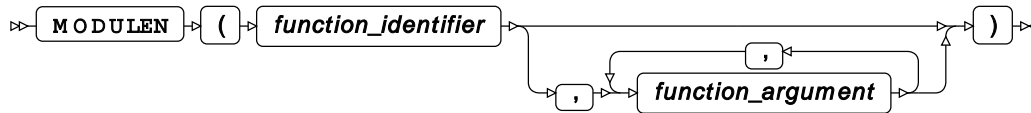
This creates an attribute table as a source element in the COPYFILE entry of the ATTRIBUTES catalog in the WORK library. This catalog entry is then accessed by the subsequent DATA step.

Using a catalog to store an attribute table that is created when it is needed ensures that the attribute table entries are available when the program is run, and provides a clean-up facility when the WPS session is ended.

MODULEN

Runs an external module that returns a numeric value.





This function must be used with modules that return numeric values. If a module returns a character, or returns no value, an error is returned. To run a module that returns a character, use [MODULEC](#) (page 1428); to run a module that returns no value, use [CALL MODULE](#) (page 1431).

Return type: Numeric

option-string

Type: Character

A character, preceded by *, that specifies the type of information returned to the log:

***E**

Error messages.

***H**

A short help text.

***I**

Detailed information about the parameters specified to the module, and the values of those parameters in both text and hexadecimal.

***T**

Summary information about the definition of the parameters in the attributes table for this module.

function_identifier

Type: Character

The name of the module (function) to be called.

function_argument

Optional argument

Type: Character or numeric value

The value required by an argument or option to a module.

The module name specified by *function_identifier* must have a corresponding attribute table, unless it is a module on z/OS; see [The attribute table](#) (page 1420) for details

Basic example

In this example, the Windows CopyFileA function is executed. The parameters required by CopyFileA are first assigned to an attribute table in the catalog entry SASCBTBL using a DATA step. The result of the return value is written to the log.

```
FILENAME SASCBTBL CATALOG "work.attributes.copyfile.source";
DATA _NULL_;
  FILE SASCBTBL;
  PUT 'routine CopyFileA';
  PUT 'module = KERNEL32';
  PUT 'minarg = 3';
  PUT 'maxarg = 3';
  PUT 'stackpop = called';
  PUT 'returns = ushort';
  PUT ' ';
  PUT 'arg 1 input char format=$cstr200.';
  PUT 'arg 2 input char format=$cstr200.';
  PUT 'arg 3 input num format=pib4. byvalue';
RUN;

DATA _NULL_;

  rc = MODULEN("E", "CopyFileA", "c:\temp\test.txt", "c:\temp\test2.txt",0);

  msg = IFC(rc, "The file copied successfully", "The file failed to copy");
  PUT msg;

RUN;
```

This produces the following output:

```
The file copied successfully
```

The third parameter in this example is set to 0 (zero). For CopyFileA, this enables the copied file to overwrite any existing file. If the parameter had been set to 1, and the file already existed, the copied file would not be able to overwrite the existing file, and the DATA step would return the message The file failed to copy.

Example – attribute table stored in file and detailed information returned

In this example, the Windows CopyFileA function is executed. The required attributes table has been already created and saved in the file c:\temp\attributes. The result of the return value, and of setting *option-string* to *I, is written to the log.

```
FILENAME SASCTBL "c:\temp\attributes";
DATA _NULL_;

  rc = MODULEN("E", "CopyFileA", "c:\temp\test.txt", "c:\temp\test2.txt",0);

  msg = IFC(rc, "The file copied successfully", "The file failed to copy");
  PUT msg;

RUN;
```

This produces the following output:

```

---PARAM LIST FOR MODULEN ROUTINE---
CHR PARM 1 0000000052F5A620 2A49 (*I)
CHR PARM 2 0000000052F5A1C0 436F707946696C6541 (CopyFileA)
CHR PARM 3 000000004C891170 633A5C74656D705C746573742E747874 (c:\temp\test.txt)
CHR PARM 4 000000004C890C30 633A5C74656D705C74657374322E747874 (c:\temp\test2.txt)
NUM PARM 5 000000005316EFD8 0000000000000000 (0.00)
---ROUTINE CopyFileA LOADED AT ADDRESS 00000000158CA370 (PARMLIST AT
 000000004C70AF20)---
PARAM 1 000000004C89A1B0
 633A5C74656D705C746573742E7478740000000000000000F0000000000000000464646
464646464646464646464646460000000000000000F0000000000000000330000004135463200516F4C1D02
000000A1894C
1D020000000000000000000000F00000000000000000000FF070080005FA5F2FC7F00007002EF521D02
000000000000
000000000F000000000000000805DA5F2FC7F00008C1E37890010008078F0CEF6FC7F000000000000FC7F
000001000000
070000000010000000A000000000000001D020000
PARAM 2 000000004C89A278
 633A5C74656D705C74657374322E7478740000000000000000000000000000
00000000037
30221B00000000000000000000F0000000000000031000000000000000656469756D0000000000000000
000000000000
000000000F0000000000000000100000000000000656469756D0000483817F3FC7F0000000000000000
00000F000000
00000000F0A2894C1D020000981E23891D11008078F0CEF6FC7F00000000000000000000007261706846
696E616C004C
1D020000000000000000000000F000000000000000
PARAM 3 00000000 <CALL-BY-VALUE>
---VALUES UPON RETURN FROM CopyFileA ROUTINE---
PARAM 1 000000004C89A1B0
PARAM 2 000000004C89A278
PARAM 3 00000000 <CALL-BY-VALUE>
---VALUES UPON RETURN FROM MODULEN ROUTINE---
CHR PARM 3 000000004C891170 633A5C74656D705C746573742E747874 (c:\temp\test.txt)
CHR PARM 4 000000004C890C30 633A5C74656D705C74657374322E747874 (c:\temp\test2.txt)
NUM PARM 5 000000005316EFD8 0000000000000000 (0.00)

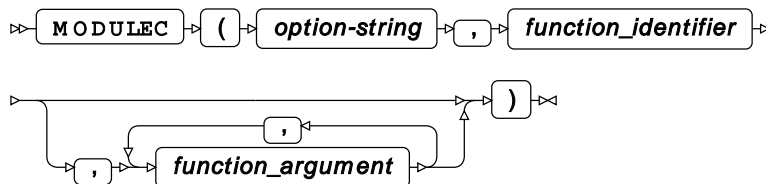
The file copied successfully

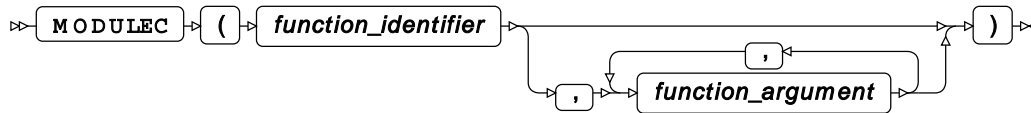
```

Because `*I` has been specified, detailed information is returned about the parameters.

MODULEC

Runs an external module that returns a character value.





This function must be used with modules that return character values. If a module returns a numeric value or returns no value, an error is returned. To run a module that returns a numeric value, use [MODULEN](#) (page 1428); to run a module that returns no value, use [CALL MODULE](#) (page 1431).

Return type: Character

option-string

Type: Character

A character, preceded by * (asterisk), that specifies the type of information returned to the log:

***E**

Error messages.

***H**

A short help text.

***I**

Detailed information about the parameters specified to the module, and the values of those parameters, in both text and hexadecimal.

***T**

Summary information about the definition of the parameters in the attributes table for this module.

function_identifier

Type: Character

The name of the module (function) to be called.

function_argument

Optional argument

Type: Character or numeric value

The value required by an argument or option to a module.

The module name specified by *function_identifier* must have a corresponding attribute table, unless it is a module on z/OS; see [The attribute table](#) (page 1420) for details

Basic example

In this example, the C++ program `createNewFileTrueFalse` is executed. It is contained in a `.dll` called `wpsExampledll`. The `createNewFileTrueFalse` program creates a specified file and returns `T` or `F`, depending on whether the file was created, and whether it overwrote an existing file.

The parameters required by `createNewFileTrueFalse` program are first assigned in an attribute table in a catalog entry `SASCBTBL` created in a `DATA` step. The result of the return value is written to the log. In this example, the `.dll` is assumed to be in the same folder as the WPS program, or in a folder specified on the `PATH` environment variable.

```
FILENAME SASCBTBL CATALOG "work.attributes.copyfile.source";
DATA _NULL_;
  FILE SASCBTBL;
  PUT 'ROUTINE createNewFileTrueFalse';
  PUT 'MODULE = wpsExampledll';
  PUT 'MINARG = 2';
  PUT 'MAXARG = 2';
  PUT 'CALLSEQ = byvalue';
  PUT 'STACKPOP = called';
  PUT 'RETURNS = char1';
  PUT ' ';
  PUT 'ARG 1 INPUT CHAR REQUIRED FORMAT=$CSTR200.';
  PUT 'ARG 2 INPUT CHAR FORMAT=$CSTR1.';
RUN;

DATA _NULL_;

  rt = MODULEEC("*E", "createNewFileTrueFalse", "c:\temp\test3.txt", "Y");
  PUT rt=;

RUN;
```

This produces the following output:

```
rt=T
```

The second parameter in this example is set to `Y`. This enables a new file to overwrite an existing file with the same name.

Example – attribute table stored in file and detailed information returned

In this example, the C++ program `createNewFileTrueFalse` is executed. It is contained in a `.dll` called `wpsExampledll`. The `createNewFileTrueFalse` program creates a specified file and returns `T` or `F`, depending on whether the file was created, and whether it overwrote an existing file.

The parameters required by `createNewFileTrueFalse` program are first assigned in an attribute table in a catalog entry `SASCBTBL` created in a `DATA` step. The result of the return value is written to the log. In this example, the `.dll` is assumed to be in the same folder as the WPS program, or in a folder specified on the `PATH` environment variable.

The required attributes table has already been created and saved in the file `c:\temp\attributes`. The result of the return value, and of setting *option-string* to `*I`, is written to the log.

```
FILENAME SASCBTBL "c:\temp\attributes";
DATA _NULL_;

    rt = MODULEC("I", "createNewFileTrueFalse", "c:\temp\test9.txt", "N");

    PUT rt=;

RUN;
```

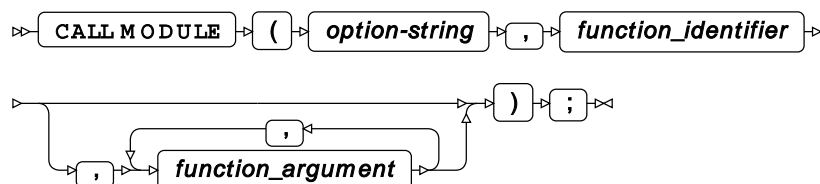
This produces the following output:

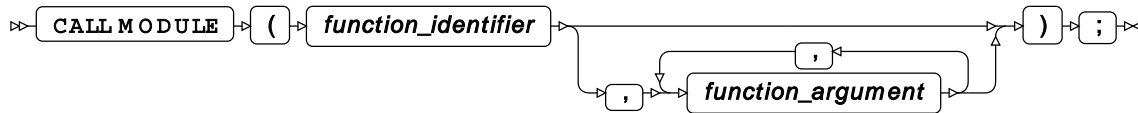
[illegible]

Because `*I` has been specified, detailed information is returned about the parameters.

CALL MODULE

Runs an external module that does not return a value.





This function must be used with modules that do not return a value. To run a module that returns a numeric value, use [MODULEN](#) (page 1428); to run a module that returns a character, use [MODULEC](#) (page 1428).

Note:

You can use this routine to run a module or program that returns a number or character, but you cannot access that return value.

option-string

Type: Character

A character, preceded by * (asterisk), that specifies the type of information returned to the log:

***E**

Error messages.

***H**

A short help text.

***I**

Detailed information about the parameters specified to the module, and the values of those parameters, in both text and hexadecimal.

***T**

Summary information about the definition of the parameters in the attributes table for this module.

function_identifier

Type: Character

The name of the module (function) to be called.

function_argument

Optional argument

Type: Character or numeric value

The value required by an argument or option to a module.

The module name specified by *function_identifier* must have a corresponding attribute table, unless it is a module on z/OS; see [The attribute table](#) (page 1420) for details

Basic example

In this example, the C++ program `createNewFile` is executed. It is contained in a `.dll` called `wpsExamp1ed11`. The `createNewFile` program creates a specified file. No value is returned.

The parameter required by `createNewFile` program is first defined in an attribute table in a catalog entry `SASCBTBL`, created in a `DATA` step. In this example, the `.dll` is assumed to be in the same folder as the WPS program, or in a folder specified on the `PATH` environment variable.

```
FILENAME SASCBTBL CATALOG "work.attributes.copyfile.source";
DATA _NULL_;
  FILE SASCBTBL;
  PUT 'ROUTINE createNewFile';
  PUT 'MODULE = wpsExamp1ed11';
  PUT 'MINARG = 1';
  PUT 'MAXARG = 1';
  PUT 'STACKPOP = called';
  PUT ' ';
  PUT 'ARG 1 INPUT CHAR REQUIRED FORMAT=$CSTR200.';
;
RUN;

DATA _NULL_;

  CALL MODULE(*I, "createNewFile", "c:\temp\mynewfile");

RUN;
```

Example – attribute table stored in file and detailed information returned

In this example, the C++ program `createNewFile` is executed. It is contained in a `.dll` called `wpsExamp1ed11`. The `createNewFile` program creates a specified file. No value is returned.

The parameter required by `createNewFile` program is first defined in an attribute table in a catalog entry `SASCBTBL`, created in a `DATA` step. In this example, the `.dll` is assumed to be in the same folder as the WPS program, or in a folder specified on the `PATH` environment variable.

The required attributes table has already been created and saved in the file `c:\temp\attributes`. The result of the return value, and of setting *option-string* to `*I`, is written to the log.

```
filename SASCBTBL "c:\temp\attributes";
DATA _NULL_;

  CALL MODULE(*I, "createNewFile", "c:\temp\mynewfile");

RUN;
```

This produces the following output:

```
---PARM LIST FOR MODULE ROUTINE---
CHR PARM 1 00000000C4FE35A0 2A49 (*I)
CHR PARM 2 00000000C4FE4830 6372656174654E657746696C65 (createNewFile)
CHR PARM 3 00000000BDDF4560 633A5C74656D705C6D796E657766696C65 (c:\temp\mynewfile)
---ROUTINE createNewFile LOADED AT ADDRESS 000000001A7C1000 (PARMLIST AT
00000000BAF73E10)---
```

[illegible]

Because `*I` has been specified, detailed information is returned about the parameters.

Example – COBOL example on z/OS

In this example, two COBOL programs are executed. The programs reads data from a dataset, which is then returned in variables to the calling WPS program.

Because the programs are COBOL programs on z/OS, no attribute table is required.

```
DATA _NULL_;
    company_name $40;
    short_name $20;
    init_name $10;
    abv_name $03;

    CALL MODULE('*EI', 'utlpname', delim, company_name, short_name, init_name,
abv_name)

    PUT company_name short_name init_name abv_name;

    CALL MODULE('*EI', 'utlpnum', company_num, short_num)

    PUT company_name short_name company_num short_num;
```

This produces the following output:

Magnificent Bicycles MagBikes MB MB1
207 27

Financial functions

Get information about various kinds of financial transactions, such as investments, assets, risk and so on.

These functions provide information on financial transactions and investments, such as:

- The future value of investments
- The total amount paid for loans at specified interest rates
- The performance of bonds and treasury bills

- Depreciation of assets
- Exposure to risk

BLACKCLPRC ↗	1437
Returns the price for a European call option using Black's model.	
BLACKPTPRC ↗	1438
Returns the price for a European put option using Black's model.	
BLKSHCLPRC ↗	1440
Returns the price for a European call option using the Black-Scholes model.	
BLKSHPTPRC ↗	1441
Returns the price for a European put option using the Black-Scholes model.	
COMPOUND ↗	1442
Returns information about the compounding of a principal.	
CONVX ↗	1445
Returns the convexity for a series of cash flows.	
CONVXP ↗	1446
Returns the convexity of a bond.	
CUMIPMT ↗	1447
Returns the cumulative interest paid over a specified period for a constant payment loan or investment.	
CUMPRINC ↗	1449
Returns the cumulative price for a loan over a specified time period.	
DACCCDB ↗	1451
Returns the accumulated declining-balance depreciation, based on a depreciation rate you specify.	
DACCDBSL ↗	1452
Returns the accumulated declining-balance depreciation using double-declining and straight-line methods.	
DACCSL ↗	1453
Returns the accumulated depreciation on an asset using a straight-line method.	
DACCSYD ↗	1454
Returns the accumulated depreciation using the sum of the years' digits method.	
DACCTAB ↗	1455
Returns the accumulated depreciation on an asset using a set of specified depreciation rates.	
DEPDB ↗	1456
Returns the declining-balance depreciation of an asset for a particular period in a specified number of periods, using a specified depreciation rate.	
DEPDBSL ↗	1458
Returns the depreciation of an asset using both declining and straight-line methods, for an asset at a particular period in a specified number of periods, using a specified declining balance depreciation rate.	

DEPSL ↗	1459
Returns the straight-line depreciation on an asset for a particular period in a specified number of periods.	
DEPSYD ↗	1460
Returns the depreciation of an asset for a particular year in a specified number of years, using the sum of the years' digits method.	
DEPTAB ↗	1461
Returns the depreciation of an asset for a particular period, using a set of depreciation rates specified for each period.	
DUR ↗	1462
Returns the modified duration for a specified set of cash flows.	
DURP ↗	1463
Returns the modified duration for a periodic cashflow, such as that generated by a bond.	
EFFRATE ↗	1464
Returns the effective interest rate.	
FINANCE ↗	1465
GARKHCLPRC ↗	1549
Returns the price for a European call option using the Garman-Kohlhagen model.	
GARKHPTPRC ↗	1551
Returns the price for a European put option using the Garman-Kohlhagen model.	
INTRR ↗	1552
Returns the internal rate of return for a supplied series of periodic cashflows.	
IPMT ↗	1553
Returns the interest paid for a specific period of a loan or investment that is being paid with constant periodic payments and has a constant interest rate.	
IRR ↗	1555
Returns the internal rate of return for a supplied series of periodic cashflows.	
MARGRCLPRC ↗	1556
Returns the price for a European call option using Margrabe's formula.	
MARGRPTPRC ↗	1557
Returns the price for a European put option using Margrabe's formula.	
MORT ↗	1558
Returns information about the amortisation of a loan.	
NETPV ↗	1560
Returns the net present value of an investment, based on a specified discount rate.	
NOMRATE ↗	1562
Returns the nominal annual interest rate.	
NPV ↗	1563
Returns the net present value of an investment, based on a specified discount rate. The rate is specified as a percentage.	

PMT ↗	1565
Returns the periodic payment necessary to pay off a loan, where the interest rate is constant.	
PPMT ↗	1567
Returns the payment necessary to repay the principal of a loan or to pay into an investment for a specified period, where the interest rate is constant and periodic payments are made.	
PVP ↗	1569
Returns the present value of a future amount, where the initial capital is repaid at maturity.	
SAVING ↗	1570
Returns the amount that would be saved based on a specified interest rate, term and payments.	
SAVINGS ↗	1572
Returns the value of savings based on consistent deposits.	
TIMEVALUE ↗	1574
Returns the value of interest on savings plus the savings if interest is applied at a specified future date while the term of the principal remains the same.	
YIELDP ↗	1577
Returns the yield to maturity for a security for specified periodic cash flows.	

BLACKCLPRC

Returns the price for a European call option using Black's model.

⇒ **BLACKCLPRC** ⇒ (*exercise-price* , *time* , *future-price* , *rate* , *volatility*) ⇒

Return type: Numeric

exercise-price

Type: Numeric

The exercise or current price for the call option.

time

Type: Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

future-price

Type: Numeric

The assumed strike price for the option.

rate**Type:** Numeric

An annualised, continuously compounded, risk-free rate of return over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100.

volatility**Type:** Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European call option that will expire in four months. The exercise price and future price are both €20; the volatility is 25% per annum, and the risk-free rate is 9% per annum. The result is written to the log.

```
DATA _NULL_;  
  cp=BLACKCLPRC(20, 0.333, 20, 0.09, 0.25);  
  PUT 'The call price is: ' cp euro5.2;  
RUN;
```

This produces the following output:

```
The call price is: E1.12
```

You could also enter *time* as a fraction:

```
cp=blackclprc(20,4/12,20,0.09,0.25);
```

This would return the same result.

BLACKPTPRC

Returns the price for a European put option using Black's model.

⇒ **BLACKPTPRC** ⇒ (*exercise-price* , *time* , *future-price* , *rate* , *volatility*) ⇒

Return type: Numeric**exercise-price****Type:** Numeric

The exercise or current price for the call option.

time**Type:** Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

future-price**Type:** Numeric

The assumed strike price for the option.

rate**Type:** Numeric

An annualised, continuously compounded, risk-free rate of return over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100.

volatility**Type:** Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European put option that will expire in four months. The exercise price and future price are both €20; the volatility is 25% per annum, and the risk-free rate is 9% per annum. The result is written to the log.

```
DATA _NULL_;  
  pp = BLKPTPRC(20, 0.333, 20, 0.09, 0.25);  
  PUT 'The put price is: ' pp euro5.2;  
RUN;
```

This produces the following output:

```
The put price is: E1.12
```

You could also enter *time* as a fraction:

```
pp=blackptprc(20,4/12,20,0.09,0.25);
```

This would return the same result.

BLKSHCLPRC

Returns the price for a European call option using the Black-Scholes model.

⇒ **BLKSHCLPRC** ⇒ (*exercise-price* , *time* , *share-price* , *rate* , *volatility*) ⇒

Return type: Numeric

exercise-price

Type: Numeric

The exercise or current price for the call option.

time

Type: Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

share-price

Type: Numeric

The assumed strike price for the option.

rate

Type: Numeric

An annualised, continuously compounded, risk-free rate of return over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100.

volatility

Type: Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European call option that will expire in four months. The exercise price and future price are both €20; the volatility is 25% per annum, and the risk-free rate is 9% per annum. The result is written to the log.

```
DATA _NULL_;  
  cp=BLKSHCLPRC(20, 0.333, 20, 0.09, 0.25);  
  PUT 'The call price is: ' cp euro5.2;  
RUN;
```

This produces the following output:

```
The call price is: E1.45
```

You could also enter *time* as a fraction:

```
cp=BLKSHCLPRC(20, 4/12, 20, 0.09, 0.25);
```

This would return the same result.

BLKSHPTPRC

Returns the price for a European put option using the Black-Scholes model.

⇒ **BLKSHPTPRC** ⇒ (*exercise-price* , *time* , *share-price* , *rate* , *volatility*) ⇒

Return type: Numeric

exercise-price

Type: Numeric

The exercise or current price for the call option.

time

Type: Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

share-price

Type: Numeric

The assumed strike price for the option.

rate

Type: Numeric

An annualised, continuously compounded, risk-free rate of return over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100.

volatility

Type: Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European put option that expires in four months. The exercise price and future price are both €20; the volatility is 25% per annum, and the risk-free rate is 9% per annum. The result is written to the log.

```
DATA EXAMPLE;  
  pp=BLKSHPTPRC(20, 0.333, 20, 0.09, 0.25);  
  PUT 'The put price is: ' pp euro5.2;  
RUN;
```

This produces the following output:

```
The put price is: E0.86
```

You could also enter *time* as a fraction:

```
pp=BLKSHPTPRC(20, 4/12, 20, 0.09, 0.25);
```

This would return the same result.

COMPOUND

Returns information about the compounding of a principal.

⇒ **COMPOUND** ⇒ (*p* , *f* , *i* , *n*) ⇐

You can calculate one of the following:

- The principal
- The payment at each period
- The interest rate
- The period

For example, if you know the principal amount, the future amount, and the number of years for which the principal is compounded, you can determine the interest rate applied. See the sections below for examples of usage.

The interest is compounded periodically (that is, every month, or every year).

Return type: Numeric

p

Type: Numeric

The principal amount to be compounded.

f

Type: Numeric

The future amount after compound interest has been applied.

i

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

n

Type: Numeric

The number of periods over which the interest is compounded.

To obtain the value of any variable in the function, you omit the variable. The value of that variable is then returned by the function. For example, to obtain the future value, you would specify:

```
ret=compound(3000,,0.05,5);
```

Or, to calculate the interest rate required to increase £3000 to £3200 over five years, you would use the following

```
ret=compound(3000,3200,,5);
```

The omitted arguments can be represented by nulls, or by missing values:

```
compound(3000,3200,,5);
```

is equivalent to:

```
compound(3000,3200,,5);
```

Example – calculating the future amount

In this example, the function calculates the future amount after an interest rate of 2% has been applied to the principal of £3000 for a number of years. The result is written to the log.

```
DATA _NULL_;  
  amount=COMPOUND(3000,,2/100,5);  
  PUT 'The amount after compounding is: ' amount nlmnlgbp10.0;  
RUN;
```

This produces the following output:

```
The amount after compounding is: £3,312.24
```

Example – calculating the principal amount

In this example, the function calculates the principal required to return a future amount of £3312 after compounding for five years at an interest rate of 2%. The result is written to the log.

```
DATA _NULL_
  pp=COMPOUND(.,3312.24,2/100,5);
  PUT 'The principal is: ' pp nlmnlgbp10.0;
RUN;
```

This produces the following output:

```
The principal is: £3,000.00
```

Example – calculating the interest rate

In this example, the function calculates the interest rate that would return a future amount of £3312 on a principal of £3,000 after five years. The result is written to the log.

```
DATA _NULL_;
  interest=compound(3000,3312,.,5);
  PUT 'The interest rate is: ' interest percent7.2;
RUN;
```

This produces the following output:

```
The interest rate is: 2.00%
```

Example – calculating the number of years

In this example, the function calculates the number of years required to increase a principal of £3,000 to a future value of £3312 if the interest rate is 2%. The result is written to the log.

```
DATA _NULL_;
  time=compound(3000,3312,2/100,);
  PUT 'The number of months required to return the future amount is: ' time 4.;
RUN;
```

This produces the following output:

```
The number of years required to return the future amount is: 5
```

Example – calculating the number of months

In this example, the function calculates the number of months required to increase a principal of £3,000 to a future value of £3312 if the interest rate is 2%. To do this, it also makes use of the `NOMRATE` function to calculate the nominal interest rate for monthly repayments. The result is written to the log.

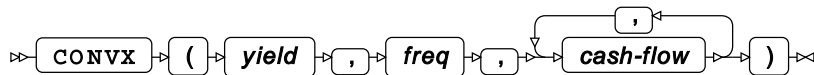
```
DATA _NULL_;
  time=COMPOUND(3000,3312,nomrate("month",2)/12,);
  PUT 'The number of months required to return the future amount is: ' time 4.;
RUN;
```

This produces the following output:

```
The number of months required to return the future amount is:    60
```

CONVX

Returns the convexity for a series of cash flows.



Return type: Numeric

yield

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, 5/100), or as a decimal (for example, 0.05).

freq

Type: Numeric

The number of coupons per period. This might, for example, be one a year, or one a month, or twelve a year.

cash-flow

Type: Numeric

A cashflow for a period. Cashflows should be entered as negative numbers if they are outgoings. Typically, the first number entered for a cashflow will be an outgoing.

Example

In this example, the convexity is calculated for a series of cash flows for a yield of 2% where the payment frequency is twice a year. The result is written to the log.

```
DATA _NULL_;
  cvx = CONVX(2/100,2,-1000,250,100,104,100,104,200,250,100,104,100,104,106);
  PUT 'The convexity is: ' cvx 7.2;
RUN;
```

This produces the following output:

```
The convexity is:    49.55
```

CONVXP

Returns the convexity of a bond.

⇒ CONVXP (*par-value* , *rate* , *numeric* , *K* , *k0* , *yield*) ⇐

Return type: Numeric

par-value

Type: Numeric

The par value of the bond.

rate

Type: Numeric

The coupon rate for the bond.

numeric

Type: Numeric

The number of coupons per period. This might, for example, be one a year, or one a month, or twelve a year.

K

Type: Numeric

The number of remaining coupon payments.

k0

Type: Numeric

The time to the first coupon. This should be greater than 0 and less than $1/\text{numeric}$. For example, if coupons are monthly, and there are 28 days until the next coupon, you might enter this value as $28/30 * 1/12$, or as 0.078.

yield

Type: Numeric

The current continuously compounded yield.

Example

In this example, the convexity is calculated for a bond with a par value of £100, a coupon rate of 5%, coupon payments twice a year with four further payments until expiry, and with a yield of 3%. There are two months until the next coupon payment. The result is written to the log.

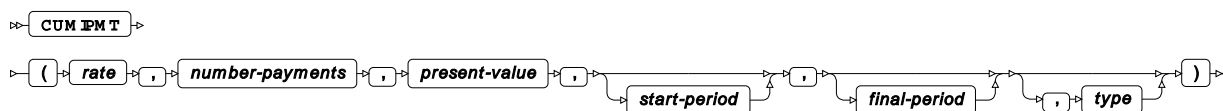
```
DATA _NULL_;
  cvx = CONVXP(100,0.05,2,4,2/6*1/2,0.03);
  PUT 'The convexity is: ' cvx 6.2;
RUN;
```

This produces the following output:

```
The convexity is:    3.32
```

CUMIPMT

Returns the cumulative interest paid over a specified period for a constant payment loan or investment.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

number-payments

Type: Numeric

The number of payments to be made into the loan or investment. For example, this might be five years, in which case you would specify 5, or 60 months, in which case you would specify 60.

present-value

Type: Numeric

The current value of the loan or investment.

start-period

Optional argument

Type: Numeric

The period within the specified number of periods at which to start calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 8. If you specify this variable, you must also specify the final period as described below.

final-period

Optional argument

Type: Numeric

The period within the specified number of periods at which to stop calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 24. If you specify this argument, you must also specify the first period as described above.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If you specify this argument, you must also specify *start-period* and *final-period*.

Basic example

In this example, the total interest is calculated for a loan whose present value is £5,000 after five years at an interest rate of 3.45%. The interest is calculated yearly. The result is written to the log.

```
DATA _NULL_;  
  ret = CUMIPMT(3.45/100, 5, 5000);  
  PUT 'The cumulative interest: ' ret 10.2 ;  
RUN;
```

This produces the following output:

```
The cumulative interest:      529.20
```

Note:

This result is the same as the total provided by summing the result of the `IPMT` [↗](#) (page 1553) function for each of the five years. See the example in the `IPMT` function.

Example – calculating the cumulative interest for a period within the term

In this example, the total interest for years four through ten is calculated for a loan of £10,000 with a term of twelve years at an interest rate of 3.45%. The result is written to the log.

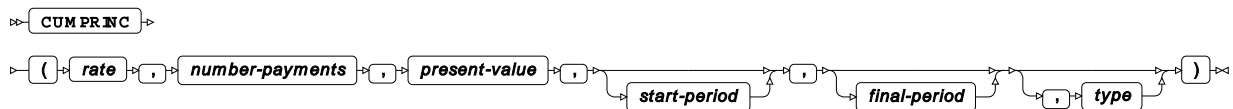
```
DATA _NULL_;
  ci = CUMIPMT((3.45/100), 12, 10000, 4,10);
  PUT 'The cumulative interest: ' ci 10.2;
RUN;
```

This produces the following output:

```
The cumulative interest:      658.19
```

CUMPRINC

Returns the cumulative price for a loan over a specified time period.



Return type: Numeric

rate

Type: Numeric

The interest rate.

number-payments

Type: Numeric

The number of payment over which the loan is repaid.

present-value

Type: Numeric

The current or final value of the loan.

start-period

Optional argument

Type: Numeric

The payment in the range *number-payments* at which to start calculating principal. For example, to start at the second payment, you would enter 2.

final-period

Optional argument

Type: Numeric

The payment in the range of *number-payments* at which to stop calculating principal. For example, to stop at the tenth payment, you would enter 10.

If omitted, the default is the final payment.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If you do not specify *start-period*, *final-period*, and *type*, the value returned will always be *present-value*.

Example

In this example, the total amount repaid over 60 months (five years) between months 24 and 40 is calculated for a loan whose final value is £15000 with an interest rate of 5%. As the period is monthly, and the rate is specified annually, the `NOMRATE` function is also used to return the appropriate nominal interest rate. The interest is calculated at the beginning of the loan periods (the default). The result is written to the log.

```
DATA _NULL_;  
  npv = CUMPRINC(NOMRATE("month",5)/12,60,15000,24,40);  
  PUT 'The cumulative principal is: ' npv nlmnlgbp12.2;  
RUN;
```

The result is written to the log.

```
The cumulative principal is: £4,266.26
```


DACCDB

Returns the accumulated declining-balance depreciation, based on a depreciation rate you specify.

➤ DACCDB ➤ (*period* , *balance* , *lifetime* , *rate*) ➤

Return type: Numeric

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The useful life of the asset, specified as a number of periods.

rate

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

If you set *period* to the same value as *lifetime*, the accumulated depreciation will equal *balance*.

Example

In this example, the accumulated value of the depreciation is calculated for years one through three of a five year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
    dac = DACCDB(3, 100000, 5,1.5);  
    PUT 'The accumulated depreciation is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The accumulated depreciation is:    65700.00
```

This is the accumulated straight-line depreciation over the first three years.

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DACCDBSL

Returns the accumulated declining-balance depreciation using double-declining and straight-line methods.

» **DACCDBSL** » (*period* , *balance* , *lifetime* , *rate*) »

The function uses a declining-balance method until the mid-point of the asset's value, and then uses a straight-line depreciation model.

For example, if your asset was worth £10,000, a declining balance method would be used until the asset was worth £5,000, after which straight line depreciation would be used.

Return type: Numeric

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The useful life of the asset, specified as a number of periods.

rate

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

If you set *period* to the same value as *lifetime*, the accumulated depreciation will equal *balance*.

Example

In this example, the accumulated value of the depreciation is calculated for years one through three of a five year depreciation on an asset worth £100,000. The declining-balance depreciation rate is set to 1.5. The result is written to the log.

```
DATA _NULL_;  
    dac = DACCDBSL(3, 100000, 5,1.5);  
    PUT 'The accumulated depreciation is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The accumulated depreciation is:    67333.33
```

This is the accumulated depreciation with declining depreciation for the first 50% of the depreciation, and straight-line depreciation afterwards.

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DACCSL

Returns the accumulated depreciation on an asset using a straight-line method.

⇒ **DACCSL** (*period* , *balance* , *lifetime*) ⇒

Return type: Numeric

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The useful life of the asset, specified as a number of periods.

If you set *period* to the same value as *lifetime*, the accumulated depreciation will equal *balance*.

Example

In this example, the accumulated value of the straight-line depreciation is calculated for years one through three of a five year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
  dac = DACCDDL(3, 100000, 5);  
  PUT 'The accumulated depreciation is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The accumulated depreciation is:    60000.00
```

Note:

Although this example has been described as applying to years, the example would be the same if the periods were months, or quarters, and so on.

DACCSYD

Returns the accumulated depreciation using the sum of the years' digits method.

➤ **DACCSYD** ➤ (*period* , *balance* , *lifetime*) ➤

Return type: Numeric

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The useful life of the asset, specified as a number of periods.

If you set *period* to the same value as *lifetime*, the accumulated depreciation will equal *balance*.

This function takes the asset's expected life specified in *lifetime*, and adds together the digits for each year. If, for example, the asset is expected to last five years, the sum of the years' digits is obtained by adding: 5 + 4 + 3 + 2 + 1 to get a total of 15. Each digit is then divided by this sum to determine the percentage by which the asset should be depreciated each year, starting with the largest; using the previous example, in the first year the depreciation would be 5/15, or 33.3%.

Example

In this example, the accumulated value of the depreciation is calculated for years one through three of a five year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
    dac = DACCDBSL(3, 100000, 5,1.5);  
    PUT 'The accumulated depreciation is: ' dac 10.2;  
RUN;
```

This produces the following output:

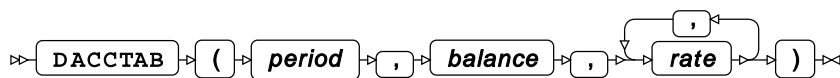
```
The accumulated depreciation is:      80000.00
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DACCTAB

Returns the accumulated depreciation on an asset using a set of specified depreciation rates.



Return type: Numeric

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

balance

Type: Numeric

The value of the asset before depreciation.

rate**Type:** Numeric

This is the rate at which the asset depreciates at a given period for a particular period.

You do not need to specify the same number of rates as there are periods; the function takes into account that the specified period might be greater than the number of fractional depreciations and interpolates. However, if you specify more rates than there are periods, only the rates required to equal the number of periods are used.

Example

In this example, the accumulated value of depreciation using a set of rates is calculated for years one through three for an asset worth £100,000.

This produces the following output:

```
DATA _NULL_;  
  y = 3;  
  dac = DEPTAB(y, 173400, 0.5, 0.25, 0.2, 0.15);  
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The depreciation in year 3 is:    34680.00
```

As only the accumulated depreciation until year three is required, the depreciation rate for year four is ignored.

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DEPDB

Returns the declining-balance depreciation of an asset for a particular period in a specified number of periods, using a specified depreciation rate.

➤ **DEPDB** ➤ (*period* , *balance* , *lifetime* , *rate*) ➤

Return type: Numeric**period****Type:** Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The number of periods over which the asset is to be depreciated. A period can represent years, or months, or quarters and so on.

rate

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

For example, if the asset is depreciating over five years, you can calculate the depreciation that will occur in the second year; if the asset is depreciating over 24 months, you can calculate the depreciation at the 15th month.

Example

In this example, the value of the depreciation is calculated for the second year of a five-year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
  y = 2;  
  dac = DEPDDB(y, 100000, 5, 1.25);  
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The depreciation in year 2 is:    18750.00
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DEPDBSL

Returns the depreciation of an asset using both declining and straight-line methods, for an asset at a particular period in a specified number of periods, using a specified declining balance depreciation rate.

➤ DEPDBSL (*period* , *balance* , *lifetime* , *rate*) ➤

Return type: Numeric

period

Type: Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The number of periods over which the asset is to be depreciated. A period can represent years, or months, or quarters and so on.

rate

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

For example, if the asset is depreciating over five years, you can calculate the depreciation that will occur in the second year; if over 17 months, you can calculate the depreciation that will occur in the eleventh month.

Example

In this example, the value of the depreciation is calculated for year two of a five-year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
  y = 2;  
  diy = DEPDBSL(y, 100000, 5, 1.25);  
  PUT 'The depreciation in year ' y 'is: ' diy 10.2;  
RUN;
```


This produces the following output:

```
The depreciation in year 2 is:    18750.00
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DEPSL

Returns the straight-line depreciation on an asset for a particular period in a specified number of periods.

```
DEPSL ( period , balance , lifetime )
```

Return type: Numeric

period

Type: Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The number of periods over which the asset is to be depreciated. A period can represent years, or months, or quarters and so on.

For example, if the asset is depreciating over five years, you can calculate the depreciation that will occur in the second year; if over 36 months, you can calculate the depreciation that will occur in the 24th month.

Example

In this example, the value of the depreciation is calculated for year four of a seven-year depreciation on an asset worth £173,500. The result is written to the log.

```
DATA _NULL_;  
  i = 4;  
  diy = DEPSL(i, 173500, 7);  
  PUT 'The depreciation in year ' i 'is: ' diy 10.2;  
RUN;
```

This produces the following output:

```
The depreciation in year 4 is:    24785.71
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DEPSYD

Returns the depreciation of an asset for a particular year in a specified number of years, using the sum of the years' digits method.

➤ **DEPSYD** (*period* , *balance* , *lifetime*) ➤

Return type: Numeric

period

Type: Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

balance

Type: Numeric

The value of the asset before depreciation.

lifetime

Type: Numeric

The number of years over which the asset is to be depreciated.

For example, if the asset is depreciating over five years, you can calculate the depreciation that will occur in the second year.

Example

In this example, the value of the depreciation is calculated for year two of a five-year depreciation on an asset worth £100,000. The result is written to the log.

```
DATA _NULL_;  
  y = 2;  
  dac = depsyd(y, 100000, 5);  
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;  
RUN;
```

This produces the following output:

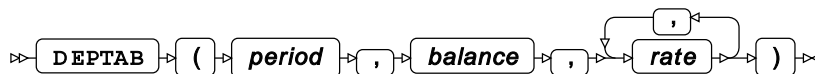
```
The depreciation in year 2 is:    26666.67
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DEPTAB

Returns the depreciation of an asset for a particular period, using a set of depreciation rates specified for each period.



Return type: Numeric

period

Type: Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

If you specify a period that is greater than *rate*, then an error message is returned. For example, if you specify 4 for *rate* but set this argument to 5, an error results.

balance

Type: Numeric

The value of the asset before depreciation.

rate

Type: Numeric

A rate of depreciation for a period.

For example, if the asset is depreciating over five years, and you have the rate for each of those years, you can calculate the depreciation that will occur in the second year.

Example

In this example, the value of the depreciation is calculated for year two where the rates for three years of depreciation have been specified. The initial value of the asset is £100,000. The result is written to the log.

```
DATA _NULL_;
  y = 2;
  dac = DEPTAB(y, 100000, 5/100, 3/100, 4/100);
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;
RUN;
```

This produces the following output:

```
The depreciation in year 2 is:      3000.00
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

DUR

Returns the modified duration for a specified set of cash flows.



Return type: Numeric

yield

Type: Numeric

The yield for the bond.

freq

Type: Numeric

The number of payments to maturity.

cash-flows

Type: Numeric

The value of a cashflow.

Example

In this example, the value of the modified duration is calculated for a bond with a yield of 10% over three years with the specified cashflows. The result is written to the log.

```
DATA _NULL_;  
  y = 3;  
  dac = DUR(10, y, 95.24, 90.70, 950.22);  
  PUT 'The modified duration is: ' dac;  
RUN;
```

This produces the following output:

```
The modified duration is: 0.07
```

DURP

Returns the modified duration for a periodic cashflow, such as that generated by a bond.

➤ **DURP** ➤ (*par-value* , *rate* , *numeric* , *K* , *k0* , *yield*) ➤

Return type: Numeric

par-value

Type: Numeric

The par value of the bond.

rate

Type: Numeric

The coupon rate for the bond.

numeric

Type: Numeric

The number of coupons per period. This might, for example, be one a year, or one a month, or twelve a year.

K

Type: Numeric

The number of remaining coupon payments.

k0

Type: Numeric

The time to the first coupon. This should be greater than 0 and less than $1/\text{numeric}$. For example, if coupons are monthly, and there are 28 days until the next coupon, you might enter this value as $28/30 * 1/12$, or as 0.078.

yield

Type: Numeric

Yield per period to maturity.

Example

In this example, the duration for a periodic cashflow is calculated for a bond with a par value of £100, a coupon rate of 5%, coupon payments twice a year with four further payments until expiry, and with a yield of 3%. There are two months until the next coupon payment. The result is written to the log.

```
dDATA _NULL_;  
  dp = DURP(100,0.05,2,4,2/6*1/2,0.03);  
  PUT 'The duration: ' dp 6.2;  
run;
```

This produces the following output:

```
The duration:      1.57
```

EFFRATE

Returns the effective interest rate.

EFFRATE (period , rate)

The effective annual interest rate is determined by the periods over which interest is calculated: daily, monthly, and so on.

Return type: Numeric

period

The period at the end of which the interest rate is applied.

"CONTINUOUS"

Interest is applied continuously.

"DAY"

Interest is applied every day.

"SEMIMONTH"

Interest is applied at the middle and end of each month.

"MONTH"

Interest is applied at the end of each month.

"QUARTER"

Interest is applied at the end of each quarter.

"SEMIYEAR"

Interest is applied at the end of semiyear (that is, half-yearly).

"YEAR"

Interest is applied at the end of each year.

rate

Type: Numeric

The interest rate, as a percentage; for example, specify 5 for 5%.

Example

In this example, the function returns the effective annual interest rate for monthly and continuous compounding. The result is written to the log.

```
DATA _NULL_;  
  eir = EFFRATE("month",5);  
  PUT 'The effective annual interest rate is: ' eir percent8.3;  
  
  eir = EFFRATE("continuous",5) * 100;  
  PUT 'The effective annual interest rate is: ' eir '%';  
  
RUN;
```

This produces the following output:

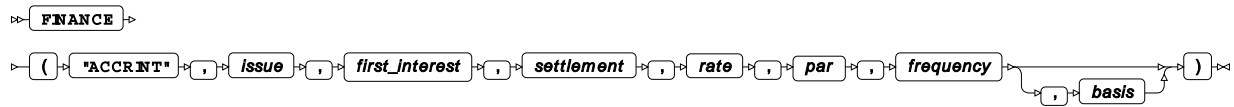
```
The effective annual interest rate is:  5.116%  
The effective annual interest rate is: 5.1271096376 %
```

FINANCE

The functions described in this section provide similar functionality to the equivalent Microsoft Excel financial functions, and enable users familiar with those functions to develop programs that will return equivalent values.

ACCRINT Calculation

Return the accrued interest for a security that pays interest at maturity.



Return type: Numeric

issue

Type: Numeric

The date on which the security was issued.

first_interest

Type: Numeric

The date on which interest will be first paid for the security.

settlement

Type: Numeric

The date on which the security will be settled.

rate

Type: Numeric

The annual coupon rate of the security.

par

Type: Numeric

The par value of the security.

frequency

Type: Numeric

The number of coupon payments per year. This must be 1, 2 or 4.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, the accrued interest for a bond with a par value of £21,000 is calculated. The coupon rate is 5%, and coupon payments are made twice a year for five years. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;
  ai = FINANCE("ACCRINT", mdy(09,11,2016), mdy(10,11,2016), mdy(09,11,2021), 5/100,
    21000, 2);
  PUT 'The accrued interest is: ' ai nlmnlgbp10.0;
RUN;
```

This produces the following output:

```
The accrued interest is: £5,250.00
```

Example – using a different day count basis

In this example, the accrued interest for a bond with a par value of £21,000 is calculated. The coupon rate is 5%, and coupon payments are made twice a year for five years. The day count basis is specified as actual/360; a month is treated as having the actual number of days, the year as 360. The result is written to the log.

```
DATA _NULL_;
  ai = FINANCE("ACCRINT", mdy(09,11,2016), mdy(10,11,2016), mdy(09,11,2021), 5/100,
    21000, 2, 2);
  PUT 'The accrued interest is: ' ai nlmnlgbp10.0;
run;
```

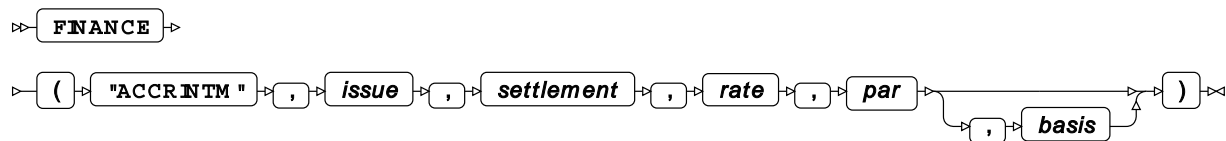
This produces the following output:

```
The accrued interest is: The accrued interest is: £5,258.75
```

Because a different day count basis has been used, the value returned is different to that returned in the previous example.

ACCRINTM Calculation

Returns the accrued interest that will be paid at maturity for a security.



Return type: Numeric

issue

Type: Numeric

The date on which the security was issued.

settlement

Type: Numeric

The date on which the security will be settled.

rate

Type: Numeric

The annual coupon rate of the security.

par

Type: Numeric

The par value of the security.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, the accrued interest at maturity for a security with a par value of £21,000 is calculated. The coupon rate is 5%, and coupon payments are made twice a year for five years. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;  
  ai = FINANCE("ACCRINTM", mdy(09,11,2016), mdy(10,11,2021), 5/100, 21000);  
  PUT 'The accrued interest is: ' ai nlmnlgbp10.0;  
RUN;
```

This produces the following output:

```
The accrued interest is: £5,337.50
```

Example – using a different day count basis

In this example, the accrued interest at maturity for a security with a par value of £21,000 is calculated. The coupon rate is 5%, and coupon payments are made twice a year for five years. The day count basis is specified as actual/360; a month is treated as having the actual number of days, the year as 360. The result is written to the log.

```
DATA _NULL_;  
  ai = FINANCE("ACCRINTM", mdy(09,11,2016), mdy(10,11,2021), 5/100, 21000,2);  
  PUT 'The accrued interest is: ' ai nlmnlgbp10.0;  
RUN;
```

This produces the following output:

```
The accrued interest is: £5,413.33
```

Because a different day count basis has been used, the value returned is different to that returned in the previous example.

AMORDEGRC Calculation

Returns the depreciation for a specified accounting period. A depreciation coefficient is applied, depending on the life of the asset. If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account. This function is provided for the French accounting system.



Return type: Numeric

cost

Type: Numeric

The cost of the asset.

date_purchased

Type: Numeric

The date on which the asset was purchased.

first_period

Type: Numeric

The date at the end of the first period.

salvage

Type: Numeric

The salvage value of the asset at the end of its life.

period

Type: Numeric

The period for which the depreciation is to be calculated.

rate

Type: Numeric

The depreciation rate.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Note:

The actual/360 day count basis is not available with this function.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, depreciation on an asset worth £50,000, with a final value of £15,000 is calculated for the fourth period. The depreciation rate is 5%. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;  
  per = 4;  
  ai = FINANCE("AMORDEGRC", 50000, mdy(09,11,2016), mdy(10,11,2016), 15000, per,  
    5/100);  
  PUT 'The amount of depreciation in period ' per 'is: ' ai nlmnlgbp10.0;  
RUN;
```

This produces the following output:

```
The amount of depreciation in period 4 is: £4,143.00
```

Example – using a different day count basis

In this example, depreciation is calculated for the fourth period for an asset worth £50,000, with a final value of £15,000. The depreciation rate is 5%. The day count basis is specified as actual/actual; that is, the months and year are treated as having the actual number of days. The result is written to the log.

```
DATA _NULL_;
  per = 4;
  ai = FINANCE("AMORDEGRC", 50000, mdy(09,11,2016), mdy(10,11,2016), 15000, per,
    5/100,1);
  PUT 'The amount of depreciation in period ' per 'is: ' ai nlmnlgbp10.0;
RUN;
```

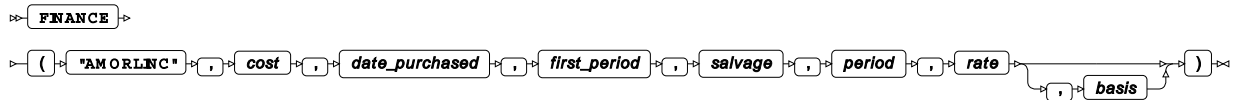
This produces the following output:

```
The amount of depreciation in period 4 is: £4,144.00
```

Because a different day count basis has been used, the value returned is different to that returned in the previous example.

AMORLINC Calculation

Returns the prorated linear depreciation for an asset for a specified accounting period.



Return type: Numeric

cost

Type: Numeric

The cost of the asset.

date_purchased

Type: Numeric

The date on which the asset was purchased.

first_period

Type: Numeric

The date at the end of the first period.

salvage

Type: Numeric

The salvage value of the asset at the end of its life.

period

Type: Numeric

The period for which the depreciation is to be calculated.

rate

Type: Numeric

The depreciation rate.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Note:

The actual/360 day count basis is not available with this function.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, depreciation on an asset worth £50,000, with a final value of £15,000 is calculated for the fourth period. The depreciation rate is 5%. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;
  per = 4;
  ai = FINANCE("AMORLINC", 50000, mdy(09,11,2016), mdy(10,11,2016), 15000, per,
    5/100);
  PUT 'The amount of depreciation in period ' per 'is: ' ai nlmnlgbp10.0;
RUN;
```

This produces the following output:

```
The amount of depreciation in period 4 is: £2,500.00
```

Example – using a different day count basis

In this example, depreciation is calculated for the fourth period for an asset worth £50,000, with a final value of £15,000. The depreciation rate is 5%. The day count basis is specified as actual/actual; that is, the months and year are treated as having the actual number of days. The result is written to the log.

```
DATA _NULL_;
  per = 13;
  ai = FINANCE("AMORLINC", 50000, mdy(09,11,2016), mdy(10,11,2017), 15000, per,
    5/100,1);
  PUT 'The amount of depreciation in period ' per 'is: ' ai nlmnlgbp10.0;
RUN;
```

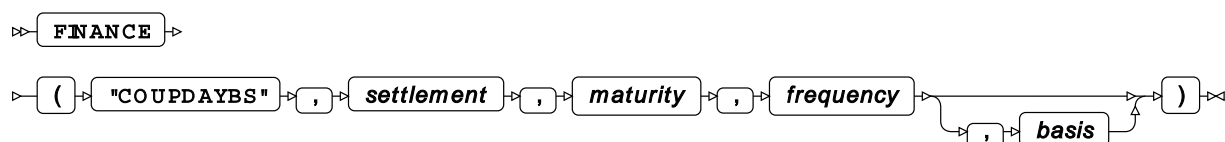
This produces the following output:

```
The amount of depreciation in period 13 is: £2,298.22
```

Because a different day count basis has been used, the value returned is different to that returned in the previous example.

COUPDAYBS Calculation

Returns, for the coupon period that contains the settlement date, the number of days from the beginning of the period until the settlement date.



For example, suppose that a ten year security matures on 06 June 2025. If coupons are paid quarterly, coupons will be paid on at 06 March, June, September and December of each year until maturity. If the security is settled on 01 January 2016, this will be 25 days after the beginning of the coupon period in which the settlement occurs, assuming a day count basis of 30/360. The coupon period would have started on 06 December 2015.

The value returned by this function depends on the day-count basis specified.

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, the number of days in the coupon period through to the settlement date is calculated for a security. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;
  days = FINANCE("COUPDAYBS", "01-Jan-2016"d, "06-Jun-2025"d, 4,0);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 25
```

Example – using a different day count basis

In this example, the number of days in the coupon period through to the settlement date is calculated for a security. The actual/actual day count basis is used. The result is written to the log.

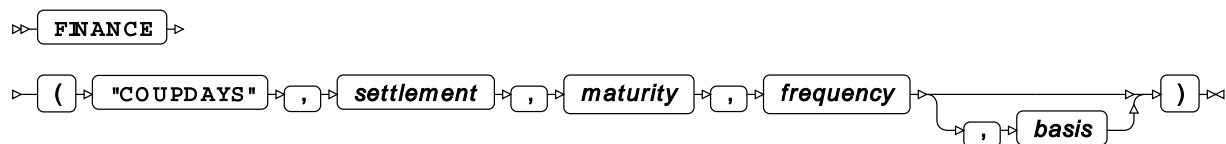
```
DATA _NULL_;
  days = FINANCE("COUPDAYBS", "01-Jan-2016"d, "06-Jun-2025"d, 4,1);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 26
```

COUPDAYBS Calculation

Returns the number of days in the coupon period containing a settlement date.



For example, suppose the settlement date is 01 January 2016 for a ten-year bond that matures on 06 June 2025. The start date for the security would, therefore, be 06 June 2015. Suppose the coupons are paid quarterly. Coupons will therefore be paid on 06 September and 06 December 2015. As the security is settled on 01 January 2016, the coupon period containing that settlement date will contain 90 days. This assumes a day count value of 30/360. The value returned will depend on the day count basis, which can be specified to the function.

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, the number of days in the coupon period containing the settlement date is calculated for a security. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;
  days = FINANCE("COUPDAYS", "01-Jan-2016"d, "06-Jun-2025"d, 2);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 180
```

Example – using a different day count basis

In this example, the number of days in the coupon period containing the settlement date is calculated for a security. The actual/actual day count basis is used. The result is written to the log.

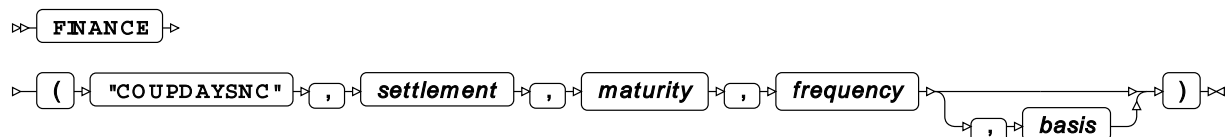
```
DATA _NULL_;
  days = FINANCE("COUPDAYS", "01-Jan-2016"d, "06-Jun-2025"d, 2,1);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 183
```

COUPDAYSNC Calculation

Returns, for the coupon period that contains the settlement date, the number of days from the settlement date to the next next coupon date.



For example, suppose the settlement date is 01 January 2016 for a ten-year bond that matures on 06 June 2025. The start date for the security would, therefore, be 06 June 2015. Suppose the coupons are paid quarterly. Coupons will therefore be paid on 06 September and 06 December 2015. As the security is settled on 01 January 2016, this will be 65 days from the settlement date until the next coupon date. This assumes a day count basis of 30/360. The value returned will depend on the day count basis, which can be specified to the function.

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Basic example

In this example, the number of days from the settlement date until the next coupon date is calculated for a security. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;
  days = FINANCE("COUPDAYSSNC", "01-Jan-2016"d, "06-Jun-2025"d, 2);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 155
```

Example – using a different day count basis

In this example, the number of days from the settlement date until the next coupon date is calculated for a security. The actual/actual day count basis is used. The result is written to the log.

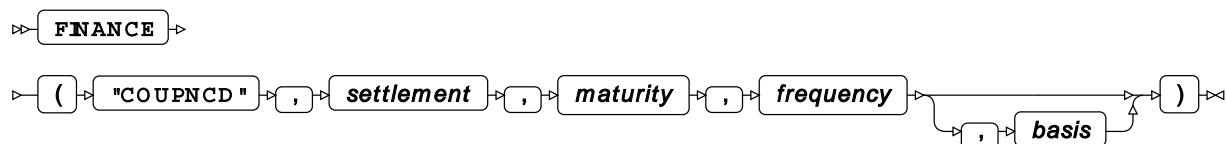
```
DATA _NULL_;
  days = FINANCE("COUPDAYSSNC", "01-Jan-2016"d, "06-Jun-2025"d, 2,1);
  PUT 'The number of days is ' days;
RUN;
```

This produces the following output:

```
The number of days is 157
```

COUPNCD Calculation

Returns the next coupon date after the specified settlement date.



For example, suppose the settlement date is 01 January 2016 for a ten-year bond that matures on 06 June 2025. The start date for the security would, therefore, be 06 June 2015. If the coupons are paid quarterly, coupons will be paid on 06 September, 06 December 2015, and 06 December 2016 (continuing quarterly thereafter). As the security is settled on 01 January 2016, the next coupon date would be 06 March 2016.

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the next coupon date from the settlement date is returned for a bond. The default day count basis is used. The result is written to the log.

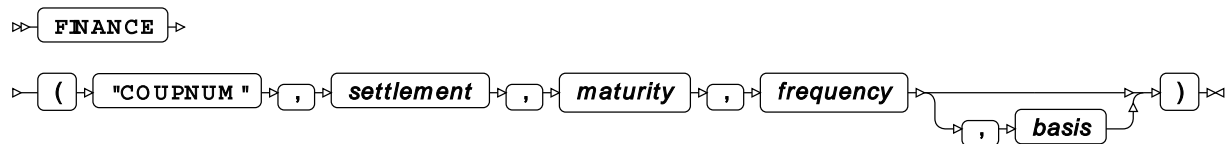
```
DATA _NULL_;  
  dt = FINANCE("COUPNCD", "01-Jan-2016"d, "28-Feb-2025"d,2);  
  PUT 'The next coupon date is ' dt date.;  
RUN;
```

This produces the following output:

```
The next coupon date is 28FEB16
```

COUPNUM Calculation

Returns the number of coupon periods remaining between the settlement date and maturity date.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the number coupon periods from the settlement date until the maturity date is calculated for a security. The default day count basis is used. The result is written to the log.

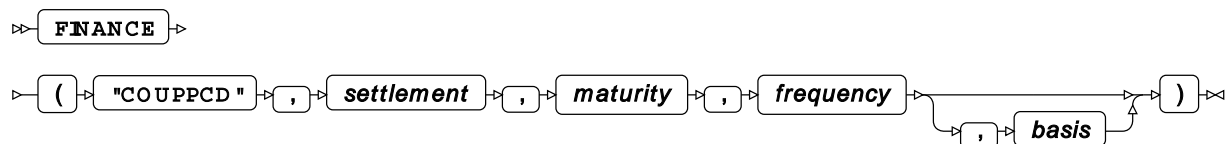
```
DATA _NULL_;
  cn = FINANCE("COUPNUM", "01-Jan-2016"d, "28-Feb-2025"d,2);
  PUT 'The number of coupons remaining is ' cn ;
RUN;
```

This produces the following output:

```
The number of coupons remaining is 19
```

COUPPCD Calculation

Returns the starting date of the last coupon period before the settlement date.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the principal will be repaid. This must be later than *settlement*, otherwise an error message is returned.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the date of the last coupon period before the settlement date is calculated for a security. The default day count basis is used. The result is written to the log.

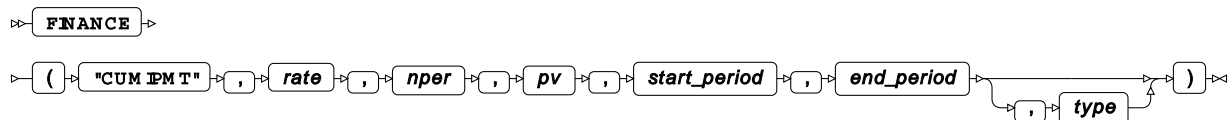
```
DATA _NULL_;
  cn = FINANCE("couppcd", "01-Jan-2016"d, "28-Feb-2025"d,4,4);
  PUT 'The date of the previous coupon period is ' cn date. ;
RUN;
```

This produces the following output:

```
The date of the previous coupon period is 28NOV15
```

CUMIPMT Calculation

Returns the cumulative interest paid over a specified period for a constant payment loan or investment.



This function is similar to the `CUMIPMT` function, except that the period for which the interest is to be calculated must be specified.

Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

nper

Type: Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

pv

Type: Numeric

The current value of the loan or investment.

start_period

Type: Numeric

The period within the specified number of periods at which to start calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 8. If you specify this variable, you must also specify the final period as described below.

end_period

Type: Numeric

The period within the specified number of periods at which to stop calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 24. If you specify this argument, you must also specify the first period as described above.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

Example

In this example, the cumulative interest is calculated for months 18 through 36 for a £10,000 loan with a term of five years (60 months). The interest rate is 3%. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("CUMIPMT", 0.03/12, 60, 10000, 18, 24);  
  PUT 'The cumulative interest is ' ret nlnmlgbp12.0;  
RUN;
```

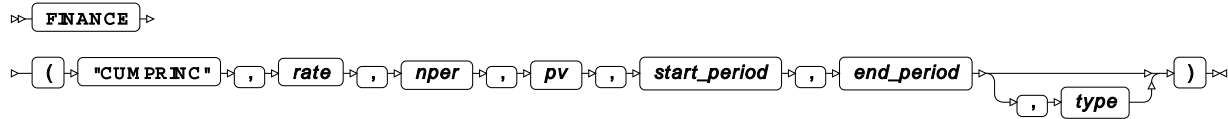
This produces the following output:

```
The cumulative interest is (£119.54)
```

As the cumulative interest is an outgoing, it is returned as a negative number (which is shown in brackets in this output format).

CUMPRINC Calculation

Returns the cumulative principal paid over a specified period for a constant payment loan or investment.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

nper

Type: Numeric

The number of payments to be made into the loan or investment. For example, this might be five years, in which case you would specify 5, or 60 months, in which case you would specify 60.

pv

Type: Numeric

The current value of the loan or investment.

start_period

Type: Numeric

The period within the specified number of periods at which to start calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 8. If you specify this variable, you must also specify the final period as described below.

end_period

Type: Numeric

The period within the specified number of periods at which to stop calculating the cumulative interest. For example, if you want to calculate the cumulative interest from the eighth month to the 24th month of a 60 month loan, you would specify 24. If you specify this argument, you must also specify the first period as described above.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If you specify this argument, you must also specify *start-period* and *final-period*.

Example

In this example, the total amount of principal repaid over five years between years two and seven is calculated for a loan that has a final value is £100,000, an interest rate of 3% and an overall repayment schedule of 10 years. Interest is applied at the beginning of the period. The result is written to the log.

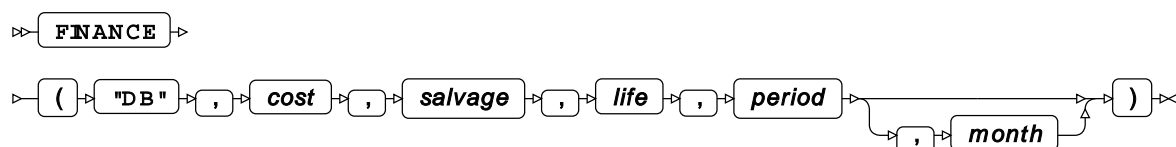
```
DATA _NULL_;
  cp = CUMPRINC(3/100, 10, 100000,2,7,1);
  PUT 'The cumulative payments are: ' cp 10.2;
RUN;
```

This produces the following output:

```
The cumulative payments are:    56424.27
```

DB Calculation

Returns the depreciation of an asset for a specified ordinal year using the fixed declining-balance method.



For example, for an asset that depreciates over five years, you can calculate the amount of depreciation that occurs in the second year.

Return type: Numeric

cost

Type: Numeric

The value of the asset before depreciation.

salvage

Type: Numeric

The value of the asset at the end of the specified number of periods.

life

Type: Numeric

The useful life of the asset, specified as a number of periods.

period

Type: Numeric

The number of periods (for example, years or months), starting from the first period for which you want the value of the accumulated depreciation. For example, if you specify 3, the calculation will apply to periods one through three.

month

Optional argument

Type: Numeric

The number of months of the year used in the calculation of the first period of depreciation. For example, if you want the calculation of depreciation to start six months after the beginning of the first year, then you set this argument to 6.

If this argument is specified, the number of months in the last period of depreciation is calculated as 12 - *month*.

If this argument is omitted, a default value of 12 is used.

The declining-balance method used by this function employs a fixed depreciation rate to calculate the amount of depreciation at the start of each period. The depreciation rate is calculated using the formula:

$$rate = 1 - (salvage/cost)^{1/life}$$

Basic example

In this example, the value of the depreciation is calculated for year two of a five-year depreciation on an asset worth £100,000, where the final value is £25,000. The result is written to the log.

```
DATA _NULL_;
  y = 2;
  dac = FINANCE("db", 100000, 25000, 5, y);
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;
RUN;
```

This produces the following output:

```
The depreciation in year 2 is:    18343.60
```

Example – depreciation begins after specified number of months

In this example, the same values are used as in the previous example; however, the calculation of depreciation doesn't begin until six months into the life of the asset. The result is written to the log.

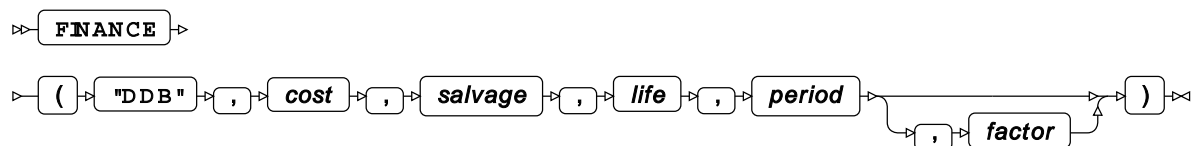
```
DATA _NULL_;
  y = 2;
  dac = FINANCE("db", 100000, 25000, 5, y, 6);
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;
RUN;
```

This produces the following output:

```
The depreciation in year 2 is:    21271.80
```

DDB Calculation

Reeturns the depreciation of an asset for a specified year. You can specify that the depreciation is calculated using a double-declining balance method, or using a specified depreciation rate.



Return type: Numeric

cost

Type: Numeric

The value of the asset before depreciation.

salvage

Type: Numeric

The value of the asset at the end of the period.

life

Type: Numeric

The useful life of the asset, specified as a number of periods.

period

Type: Numeric

The period within *life* for which you want the depreciation. For example, if you wanted the depreciation in the third year of a five-year life, you would specify 3.

factor

Optional argument

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

The default value is 2.

Basic example

In this example, the value of the depreciation is calculated for year two of a five-year depreciation on an asset worth £100,000, where the final value is £25,000. The result is written to the log.

```
DATA _NULL_;  
  y = 2;  
  dac = FINANCE("DDB", 100000, 25000, 5, y);  
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The depreciation in year 2 is:    24000.00
```

Example – using a specified depreciation rate

In this example, the same values are used as in the previous example; however, the depreciation rate is specified as 1.25%. The result is written to the log.

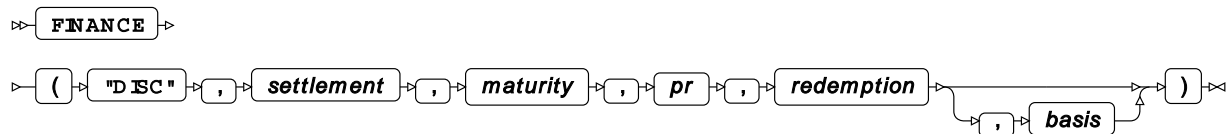
```
DATA _NULL_;  
  y = 2;  
  dac = FINANCE("DDB", 100000, 25000, 5, y, 1.25);  
  PUT 'The depreciation in year ' y 'is: ' dac 10.2;  
RUN;
```

This produces the following output:

```
The depreciation in year 2 is: 18750.00
```

DISC Calculation

Returns the discount rate for a bond.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

pr

Type: Numeric

Price of the security.

redemption

Type: Numeric

Redemption value of the security.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Example

In this example, a bond is bought for £95 and matures with a value £105 two years later. The result is written to the log.

```
DATA _NULL_;  
  dsc = FINANCE("disc", "1-Dec-2017"d, "1-Dec-2019"d, 95, 105);  
  PUT 'The discount rate for the security is:' dsc percent7.3;  
RUN;
```

This produces the following output:

```
The discount rate for the security is: 4.76%
```

DOLLARDE Calculation

Returns as a decimal value a security price originally quoted as a fractional value.

➤ **FINANCE** ➤ ("DOLLARDE" , *fractional_dollar* , *fraction*) ➤

For example, a price might be quoted as 1 and 1/16 of a particular currency; this function can be used to convert to the decimal equivalent, 1.0625.

Return type: Numeric

fractional_dollar

Type: Numeric

The fractional price of the security. For example, if the price is 1 and 1/16 of a dollar, specify 1.01; if 1 and 10/16 of a dollar, specify 1.10.

fraction

Type: Numeric

The denominator for the fraction. For example, if the price is 1 and 1/16 of a dollar, specify 16; if the price is 1 and 3/8 of a dollar, specify 8. This value should be greater than 1. If you specify 0, an error is returned.

Note:

If the value after the period in *fractional_dollar* is equal to or greater than *fraction*, the value before the period is incremented in the result. For example, if *fraction* is 16 and the *fractional_dollar* value is 1.16, the result will be 2. If *fraction* is 16 and the *fractional_dollar* value is 1.17, the result will be 2.0625.

Example

In this example, the decimal value is calculated for a security quoted as 1 and 3/16 of a dollar. The result is written to the log.

```
DATA _NULL_;  
    ret = FINANCE("DOLLARDE", 1.03, 16);  
    PUT 'The decimal value is: ' ret;  
RUN;
```

This produces the following output:

```
The decimal value is: 1.1875
```

DOLLARFR Calculation

Returns as a fractional value a security price originally quoted as a decimal value.

➤ **FINANCE** ➤ ("DOLLARFR" , *decimal_dollar* , *fraction*) ➤

For example, a price might be quoted as 1.0625. This function can be used to convert to a decimal equivalent, such as 1 and 1/16th of a dollar.

Converted values are returned in the form *n.f*, where *f* represents the numerator for a specified fractional value. For example, a price quoted as 1.0625 would be returned as 1.01 if the fractional value is 16. In this case, .01 represents 1/16. Similarly, 1.625 would be returned as 1.1, where the .1 represents 10/16.

Return type: Numeric

decimal_dollar

Type: Numeric

The decimal price of the security; for example, 1.625.

fraction

Type: Numeric

The denominator for the fraction. For example, if you want the price returned in sixteenths of a dollar, specify 16; if in eighths, specify 8. This value should be greater than 1. If you specify 0, an error is returned.

Example

In this example, the value of a security is quoted as \$1.25. The fractional value is calculated, using eighths as the appropriate fraction. The result is written to the log.

```
DATA _NULL_;  
    ret = FINANCE("DOLLARFR", 1.25, 8);  
    PUT 'The fractional value is: ' ret;  
RUN;
```

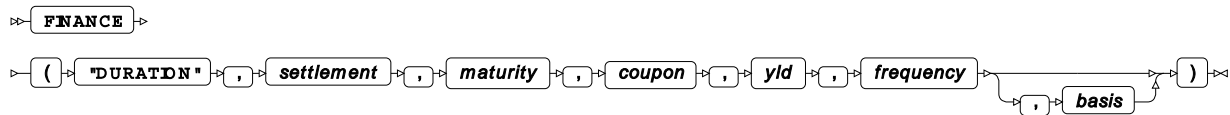
This produces the following output:

```
The fractional value is: 1.2
```

where 2 indicates 2/8 of a dollar.

DURATION Calculation

Returns the Macaulay duration of a security that pays periodic interest. The par value for the security is assumed to be \$100.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

coupon

Type: Numeric

The annual coupon rate for the security. This is a percentage and should be expressed as a fraction or decimal number; for example, 10/100 or 0.10 for 10%.

yld

Type: Numeric

The annual yield for the security. This is a percentage and should be expressed as a fraction; for example, 8/100 or 0.08 for 8%.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Basic example

In this example, the duration is calculated for a security. The default day count basis is used. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("DURATION", "01-Apr-2015"d, "31-Mar-2025"d, 10/100, 8/100,4);  
  PUT 'The duration is ' ret 7.5;  
RUN;
```

This produces the following output:

```
The duration is 6.67165
```

Example – using a different day count basis

In this example, the duration is calculated for a security. The actual/actual day count basis is used. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("DURATION", "01-Apr-2015"d, "31-Mar-2025"d, 10/100, 8/100,4,1);  
  PUT 'The duration is ' ret 7.5;  
RUN;
```

This produces the following output:

```
The duration is 6.67168
```

EFFECT Calculation

Returns the effective interest rate.

➤ **FINANCE** ➤ ("EFFECT" , *nominal_rate* , *npery*) ➤

The effective annual interest rate is determined by the periods over which interest is calculated: daily, monthly, and so on. The effective rate is returned as a percentage.

Return type: Numeric

nominal_rate

Type: Numeric

The nominal interest rate.

npery

Type: Numeric

The number of periods. For example, payments twice a year (semi-annually) should be specified as 2; payments 4 times a year (quarterly) should be specified as 4.

Example

In this example, the function returns the effective annual interest rate for semiannual compounding. The result is written to the log.

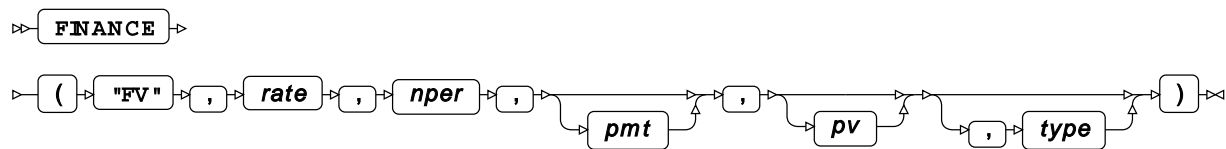
```
DATA _NULL_;  
  eir = FINANCE("effect",5/100,4);  
  PUT 'The effective annual interest rate is:' eir percent7.4;  
RUN;
```

This produces the following output:

```
The effective annual interest rate is: 5.09%
```

FV Calculation

Returns the future value of an investment that has periodic constant payments, and to which a constant interest rate is applied.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

nper

Type: Numeric

The number of periods over which payments will be made to the investment. For example, five years might be specified as 5, or as 60, for 60 months.

pmt

Optional argument

Type: Numeric

The payment amount per period.

This should be negative as it is an outgoing payment.

pv

Optional argument

The current value of the investment.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If you omit *pmt*, but want to include additional arguments, remember to include its corresponding comma, as shown in the first example below.

Example – future value of an initial investment

In this example, the function returns the future value for an initial investment of £10,000 that is compounded annually with an interest rate of 5%. The result is written to the log.

```
DATA _NULL_;  
    tfvis = FINANCE("FV", 5/100, 4,, -10000);  
    PUT 'The future value is: ' tfvis ;  
RUN;
```

This produces the following output:

```
The future value is: 12155.0625
```

In this example, there is no monthly payment, so the *pmt* parameter has been omitted. If you omit an argument, the corresponding comma should still be used to delimit it. You can also specify a missing value; for example:

```
finance("FV", 5/100, 4,., -10000);
```

Example – future value of monthly investments

In this example, the function returns the future value for monthly payments of £100 for five years that are compounded annually with a 5% interest rate. The result is written to the log.

```
DATA _NULL_;
  tfvis = FINANCE("fv", (5/100)/12, 60,-100);
  PUT 'The future value is: ' tfvis nlmnlgbp12.0 ;
RUN;
```

This produces the following output:

```
The future value is: £6,800.61
```

Example – future value with an investment and monthly payments

In this example, the function returns the future value for monthly payments of £100 for ten years, with an initial deposit of £10,000, and with an annual interest rate of 5%. In this example, payments are made at the beginning of each period. In this example, payments are made at the beginning of the period. The result is written to the log.

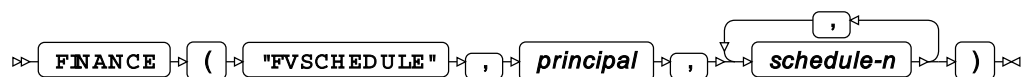
```
DATA _NULL_;
  tfvis = FINANCE("fv", (5/100)/12, 60,-100, -10000,1);
  PUT 'The future value is: ' tfvis nlmnlgbp12.0 ;
RUN;
```

This produces the following output:

```
The future value is: £19,662.53
```

FVSCCHEDULE Calculation

Returns the future value of an investment using variable interest rates over a number of periods.



Return type: Numeric

principal

Type: Numeric

The amount invested

schedule-n

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

You specify as many *schedule-n* as there are periods of interest. You can specify a different interest rate for each period; see the example below for an illustration.

Example

In this example, the function returns the future value for an initial investment of £10,000 that is compounded annually with an interest rate of 5% for two years, and an interest rate of 3.5% for the following three years. This produces the following output:

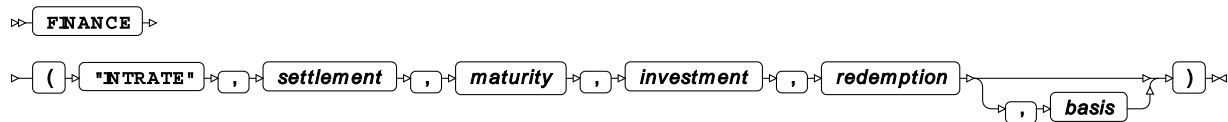
```
DATA _NULL_;
  tfvis = FINANCE("FVSCHEDULE", 10000, 5/100, 5/100, 3.5/100, 3.5/100, 3.5/100);
  PUT 'The future value is: ' tfvis nlnmlgbp12.0 ;
RUN;
```

This produces the following output:

```
The future value is: £12,223.61
```

INTRATE Calculation

Returns the interest rate for a security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

investment

Type: Numeric

The amount invested into the security.

redemption

Type: Numeric

The amount to be received at maturity.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Example

In this example, the function returns the interest rate for a security which has an initial value of £10,000 and a maturity value of £14,700. In this example, payments are made at the beginning of the period. The result is written to the log.

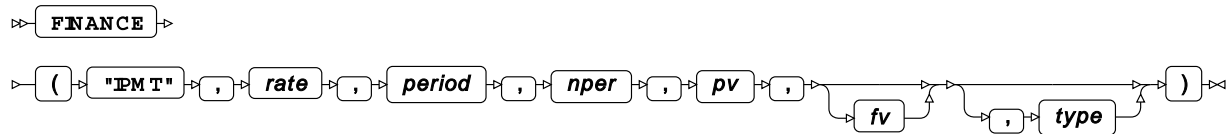
```
DATA _NULL_;  
  tiris = FINANCE("INTRATE", "01-Jan-2015"d, "2-Jun-2025"d, 10000, 14700);  
  PUT 'The interest rate is: ' tiris percent6.4 ;  
RUN;
```

This produces the following output:

```
The interest rate is: 4.5%
```

IPMT Calculation

Returns the interest paid or received for a specified period of a loan or investment that is being paid with constant periodic payments with a constant interest rate.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

period

Type: Numeric

The period for which the interest payment is to be calculated. This must be an integer between 1 and *number-payments*. For example, this might be two, in which case you would specify 2, or the 24th month, in which case you would specify 24.

nper

Type: Numeric

The number of payments to be made into the loan or investment. For example, this might be five years, in which case you would specify 5, or 60 months, in which case you would specify 60.

pv

Type: Numeric

The current value of the loan or investment.

fv

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If you specify this variable, you must also specify *start-period* and *final-period*.

Basic example

In this example, the interest paid is calculated for a loan that has an interest rate of 3.5% and a present value of £5,000. The interest to be paid is calculated for year two of a five year loan period. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;  
    tipis = FINANCE("IPMT", 3.5/100, 2, 5, 5000);  
    PUT 'The interest for the period is: ' tipis 6.4;  
RUN;
```

This produces the following output:

```
The interest for the period is: (£142.37)
```

The number returned is a negative number, indicating that this is an outgoing.

Example – investment with present and future values

In this example, the interest paid is calculated for an investment that loan that has an interest rate of 3.5%, a present value of £0 and a future value of £7,500. The interest paid is calculated for years two of the investment period. The interest is calculated at the beginning of each period. The result is written to the log.

```
DATA _NULL_;  
    tipis = FINANCE("IPMT", 3.5/100, 2, 5, 0, 7500,1);  
    PUT 'The interest for the period is: ' tipis nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The interest for the period is: £47.30
```

IRR Calculation

Returns the internal rate of return for a supplied series of periodic cashflows.



Return type: Numeric

The internal rate of return is returned as a percentage.

value

Type: Numeric

A cashflow for a period.

The first cashflow specified should be the initial investment, and should thus be entered as a negative number.

If the cash flows are non-periodic, you should use the [FINANCE XIRR](#) (page 1542) function.

Example

In this example, the internal rate of return is calculated for a supplied series of cashflows. The result is written to the log.

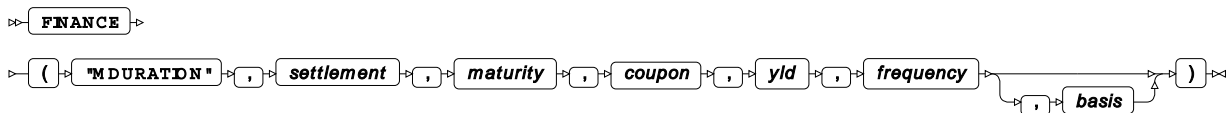
```
DATA _NULL_;
  rr = FINANCE("irr", -100, 20, 24, 28.80, 34.56, 41.47);
  PUT 'The internal rate of return is: ' rr percent7.3;
RUN;
```

This produces the following output:

```
The internal rate of return is: 13.057575638
```

MDURATION Calculation

Returns the modified Macaulay duration of a security.



Return type: Numeric

The duration is returned as the number of years.

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

coupon

Type: Numeric

The annual coupon rate.

yld

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, 5/100), or as a decimal (for example, 0.05).

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Example

In this example, the value of the modified duration for a bond with an annual yield of 8% and a coupon rate of 10% is calculated over three years with the semi-annual periods. The result is written to the log.

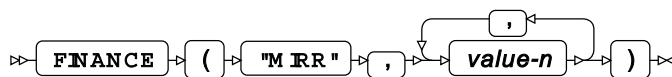
```
DATA _NULL_;
  md = FINANCE("mduration", "12-Feb-2015"d, "11-Feb-2018"d, 10/100, 8/100, 2);
  PUT 'The modified duration is: ' md ;
RUN;
```

This produces the following output:

```
The modified duration is: 2.568919923
```

MIRR Calculation

Returns the modified internal rate of return (MIRR) for a series of periodic cashflows.



The MIRR differs from the internal rate of return (IRR) by considering both the initial cost of the investment and the interest received on the reinvestment of cash.

Return type: Numeric

The value is returned as a fraction. To show this as a percentage, you can multiply the value by 100, or format it as a percentage as shown in the example below.

value-n

Type: Numeric

One of the following:

- A cashflow for a period
- The interest rate applied to the money in the cash flow (the finance rate)
- The interest rate applied to a reinvested cash flow (the reinvestment rate)

Interest rates should be entered as fractions or decimal numbers; for example, 5/100 or 0.05.

The variables supplied in *value-n* must be specified in the following order:

1. One or more cashflows. The first cashflow specified should be the initial investment, and should thus be entered as a negative number
2. The finance rate

3. The reinvestment rate

See below for an example of how to specify cash flows and rates.

Example

In this example, the internal rate of return is calculated for a supplied series of cashflows. The finance rate is 6.5%, and the reinvestment rate is 6%. The result is written to the log.

```
DATA _NULL_;  
  err = FINANCE("MIRR", -100, 15, 21.50, 18.80, 22.50, 30, 6.5/100, 6/100);  
  PUT 'The modified internal rate of return is: ' err percent8.2;  
RUN;
```

This produces the following output:

```
The modified internal rate of return is:    3.63%
```

NOMINAL Calculation

Returns the annual nominal interest rate.

⇒ **FINANCE** ⇒ ("NOMINAL", *effect_rate* , *npery*) ⇒

Return type: Numeric

effect_rate

Type: Numeric

The effective annual interest rate.

npery

Type: Numeric

The number of compounding periods in a year.

Example

In this example, the nominal interest rate is returned for an interest rate of 2.5% per annum compounded quarterly. The result is written to the log.

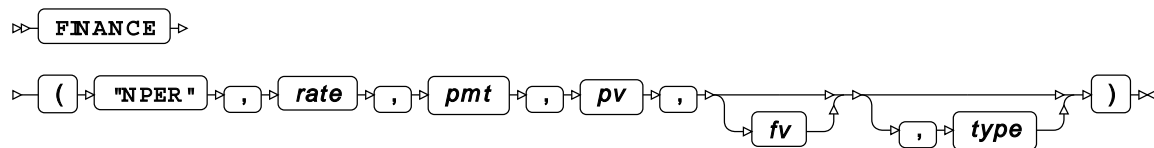
```
DATA _NULL_;  
  npv = FINANCE("nominal", 2.5/100, 4);  
  PUT 'The nominal interest rate is: ' npv percent7.2;  
RUN;
```

This produces the following output:

```
The nominal interest rate is: 2.48%
```

NPER Calculation

Returns the number of periods required to pay back a loan, or the number of payments required to save a specified amount, based on a particular interest rate.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

pmt

Type: Numeric

The payment amount per period.

As this is an outgoing amount, you should enter this as a negative number. For example if the payment is £1,000 a month, enter -1000.

pv

Type: Numeric

The current value of the loan or investment.

For a loan, this might be, for example, 5000; for savings, this might be 0.

fv

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

If you wanted to know the number of payments until a loan is paid off, you would enter 0; if you wanted to know the number of payments between an initial loan amount of £5,000 and an outstanding amount of £1,000, you would enter 1000. If you wanted to know how many payments you would need to reach savings of £10,000, you would enter 10000.

If this argument is not specified, the default value is 0 (zero).

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

Basic example

In this example, the number of payments is calculated for a loan of £10,000 repaid at £1,000 per month. The future value has been omitted, and will default to 0 (zero). The result is written to the log.

```
DATA _NULL_;  
  np = FINANCE("NPER", (2.5/100)/12, -1000, 10000);  
  PUT 'The number of repayments is: ' np;  
RUN;
```

This produces the following output:

```
The number of repayments is: 10.116159468
```

That is, about 10 months and 3 days.

Example – paying into savings

In this example, the number of payments required to reach an amount of £10,000, with an initial deposit of £2,000, paid at £1,000 per quarter, with an annual interest rate of 2.5%. The payment in this example is made at the beginning of the period. The result is written to the log.

```
DATA _NULL_;  
  np = FINANCE("NPER", (2.5/100)/4, -1000, 2000, 10000, 1);  
  PUT 'The number of repayments is: ' np;  
RUN;
```

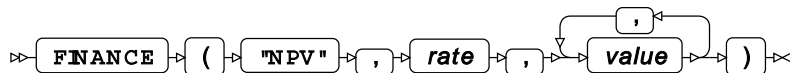
This produces the following output:

```
The number of repayments is: 11.749108338
```

That is, approximately 11 quarters and three-quarters of a quarter, or two years and three months.

NPV Calculation

Returns the net present value of an investment, based on a specified percentage discount rate.



Return type: Numeric

rate

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

value

Type: Numeric

A payment or income. If a payment, specify as a negative number; if income, specify as a positive number.

You should enter cash flows in the same sequence as they came in or were deposited. The function assumes that each cash flow occurs at the end of each period, and that each period is equal in length.

The annual discount rate might represent the rate of inflation, or it might represent the interest rate of a competing investment.

The investment begins one period before the date of the first cash flow and ends with the last cash flow in the list. The calculation is based on future cash flows; therefore, if the first cash flow occurs at the beginning of the first period, you should add this value to the result of this function, and not include it in the values.

Basic example

In this example, the net present value is calculated for an investment of £10,000 with the subsequent specified cash flows. The discount rate is 2%. In this example, there are five cash flows, so the NPV is calculated over five years. The result is written to the log.

```
DATA _NULL_;
  pv = FINANCE("NPV", 0.02, -10000, 2500, 2600, 2900, 3700, 4200);
  PUT 'The net present value is: ' pv nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The net present value is: £4,808.87
```

Example – payment at beginning of first period

In this example, the variables are the same as the previous example, except the initial payment of £10,000 is made at the beginning of the first period. The result is written to the log.

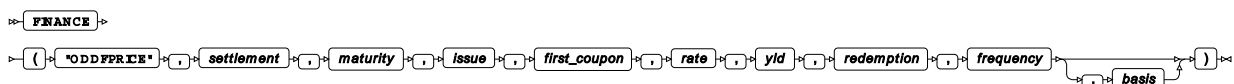
```
DATA _NULL_;
  pv = -10000 + finance("NPV", 0.02, 2500, 2600, 2900, 3700, 4200);
  PUT 'The net present value is: ' pv nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The net present value is: £4,905.05
```

ODDFPRICE Calculation

Returns the price of a security with an odd (that is, a short or long) first period. The calculation is based on a \$100 face value for the security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

issue

Type: Numeric

Date on which the security was issued. This must be earlier than *settlement*, otherwise an error message is returned.

first_coupon

Type: Numeric

Date on which the first coupon is paid. This must be earlier than *maturity* but later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as $5/100$ or as 0.05 .

yld

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, $5/100$), or as a decimal (for example, 0.05).

redemption

Type: Numeric

The redemption value, per \$100 face value.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the price of a bond with an odd first period is calculated over three years. The bond has an annual coupon rate of 5.5%, a yield of 3.5%, and pays quarterly. The result is written to the log.

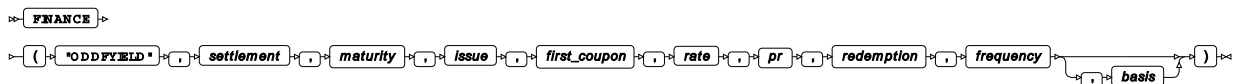
```
DATA _NULL_;
  oddp = FINANCE("ODDFPRICE", "01-Feb-2011"d, "31-Mar-2015"d, "01-Dec-2010"d,
                "31-Mar-2011"d, 5.5/100, 3.5/100, 100, 4);
  PUT 'The return is: ' oddp ;
RUN;
```

This produces the following output:

```
The return is: 107.70358549
```

ODDFYIELD Calculation

Returns the yield for a security with an odd (that is, a short or long) first period. The calculation is based on a \$100 face value for the security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

issue

Type: Numeric

Date on which the security was issued. This must be earlier than *settlement*, otherwise an error message is returned.

first_coupon

Type: Numeric

Date on which the first coupon is paid. This must be earlier than *maturity* but later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

pr

Type: Numeric

The price of the security at maturity.

redemption

Type: Numeric

The redemption value, per \$100 face value.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield of a bond with an odd first period is calculated over three years. The bond has an annual coupon rate 5.5% and a price of \$100. The bond pays quarterly. The result is written to the log.

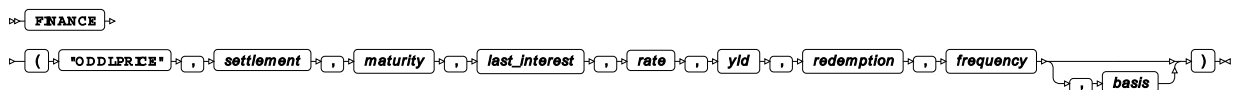
```
DATA _NULL_;
  oddyd = FINANCE("ODDFYIELD", "01-Feb-2011"d, "31-Mar-2015"d, "01-Dec-2010"d,
    "31-Mar-2011"d, 5.5/100, 95, 110, 4);
  PUT 'The return is: ' oddyd percent7.2;
RUN;
```

This produces the following output:

```
The return is: 8.96%
```

ODDLPRICE Calculation

Returns the price of a security with an odd (that is, a short or long) last period. The calculation is based on a \$100 face value for the security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

last_interest

Type: Numeric

Date on which the last coupon is paid. This must be earlier than *maturity* but later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

yld

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, 5/100), or as a decimal (for example, 0.05).

redemption

Type: Numeric

The redemption value, per \$100 face value.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the value of a bond with an odd last period is calculated. The bond has an annual coupon rate 5.5% and a yield of 3.5%. The security pays quarterly. The result is written to the log.

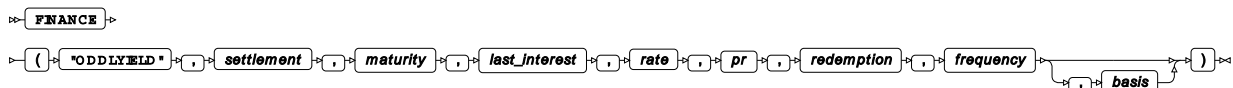
```
DATA _NULL_;
  oddp = FINANCE("oddlprice", "01-Jun-2016"d, "31-Aug-2016"d, "4-Jan-2016"d,
                5.5/100, 3.5/100, 100, 4);
  PUT 'The return is: ' oddp;
RUN;
```

This produces the following output:

```
The return is: 100.47618236
```

ODDLYIELD Calculation

Returns the yield for a security with an odd (that is, a short or long) last period. The calculation is based on a \$100 face value for the security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

last_interest

Type: Numeric

Date on which the last coupon is paid. This must be earlier than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

pr

Type: Numeric

The price of the security at maturity.

redemption

Type: Numeric

The redemption value, per \$100 face value.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield of a bond with an odd last period is calculated. The bond has an annual coupon rate 5.5%, and the price is \$99.5. The security pays quarterly. The result is written to the log.

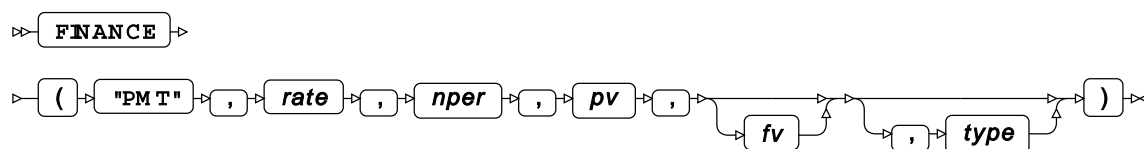
```
DATA _NULL_;
  oddp = FINANCE("oddyield", "15-Apr-2016"d, "30-Jun-2016"d, "31-Jan-2016"d,
                5/100, 99.5, 100, 4);
  PUT 'The yield is: ' oddp percent7.2;
RUN;
```

This produces the following output:

```
The yield is: 7.36%
```

PMT Calculation

Returns the periodic payment necessary to pay off a loan, where the interest rate is constant.



Return type: Numeric

rate**Type:** Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

nper**Type:** Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

pv**Type:** Numeric

The current value of the loan or investment.

fv

Optional argument

Type: Numeric

The value of the loan after *period* payments.

If this argument is not specified, the default value is 0 (zero).

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

Basic example

The following example calculates the monthly payment required to completely repay a loan over five years. The payments are made monthly. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;
  pymt = FINANCE("pmt", 0.05/12, 60, 50000);
  PUT 'The payments are: ' pymt nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The payments are: (£943.56)
```

The payment is a negative number (indicated by the brackets in this format), as it is an outgoing.

Paying for an investment

The following example calculates the quarterly payments for a five year investment with a current value of £20,000 and a future value of £50,000. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;
  pymt = FINANCE("PMT", NOMRATE("QUARTER", 0.05)/4, 5*4, 20000, 50000,1);
  PUT 'The payments are: ' pymt nlmnlgbp12.2;
RUN;
```

This produces the following output:

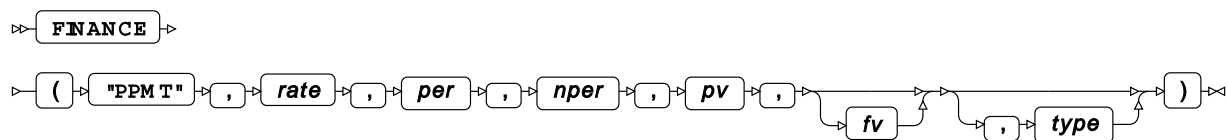
```
The payments are: (£3,497.91)
```

As the payments are made quarterly, the effective interest rate based on quarterly payments is calculated using the `EFFRATE` function. The loan period of five years needs to be multiplied by four to ensure the correct number of repayment periods is specified.

The payment is a negative number (indicated by the brackets in this format), as it is an outgoing.

PPMT Calculation

Returns the payment necessary to pay off the principal of a loan or investment for a specified period, where the interest rate is constant and periodic payments are made.



Return type: Numeric

rate**Type:** Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

per**Type:** Numeric

The period for which the interest payment is calculated. This must be an integer between 1 and *number-payments*. For example, you might want the payment for the second year in all payments, in which case you would specify 2, if the number of payments is specified in years, or 24, if the number of payments is specified in months.

nper**Type:** Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

pv**Type:** Numeric

The current value of the loan or investment.

fv

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

Example

In this example, the interest paid is calculated for a loan that has an interest rate of 5% and a present value of £5,000. The interest to be paid is calculated for each year of a five year loan period. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;
  i = 3;
  pymt = finance("PPMT", EFFRATE("monthly",12)/12, i, 60, 50000);
  PUT 'The payment for month ' i 'is : ' pymt nlmnlgbp12.2;
RUN;
```

This produces the following output:

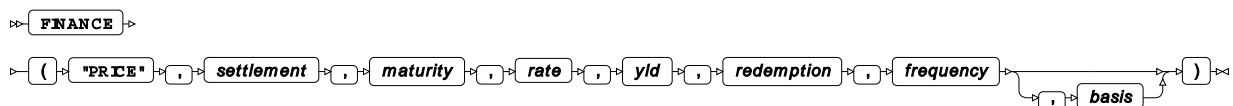
```
The payment for month 3 is : (£743.33)
```

The payment is a negative number (indicated by the brackets in this format), as it is an outgoing.

As the payments are made monthly, the effective interest rate based on monthly payments is calculated using the `EFFRATE` function. The loan period of five years is specified as 60 months.

PRICE Calculation

Returns the price of a security for which periodic interest is paid. The security is assumed to have a \$100 face value.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as $5/100$ or as 0.05 .

yld

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, $5/100$), or as a decimal (for example, 0.05).

redemption

Type: Numeric

The redemption value, per \$100 face value.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the price per \$100 face value for a security three years bond with an annual coupon rate 5.5% and a yield of 3.5%. The security pays quarterly. The result is written to the log.

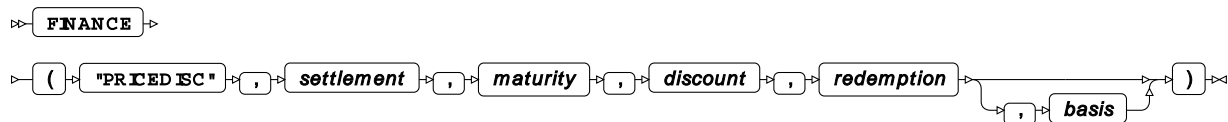
```
DATA _NULL_;
  oddp = FINANCE("price", "01-Feb-2011"d, "31-Mar-2015"d, 5.5/100, 3.5/100, 100, 4);
  PUT 'The return is: ' oddp nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The return is: £107.72
```

PRICEDISC Calculation

Returns the price of a discounted security. The security is assumed to have a \$100 face value.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

discount

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

redemption

Type: Numeric

The redemption value, per \$100 face value.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the discounted price for a bond with a value of \$110 face value for a security over three years with a discount rate of 5.5%. The security pays quarterly. The result is written to the log.

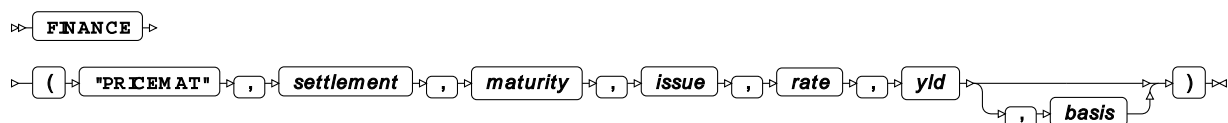
```
DATA _NULL_;
  oddp = FINANCE("pricedisc", "01-Feb-2015"d, "31-Mar-2018"d, 5.5/100, 110, 4);
  PUT 'The discount price is: ' oddp nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The discount value is: £90.86
```

PRICEMAT Calculation

Returns the price, per \$100 face value, of a security that pays interest at maturity.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

issue

Type: Numeric

The issue date of the security. This must be earlier than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as $5/100$ or as 0.05 .

yld

Type: Numeric

The security's yield, as a percentage. You can enter this as a fraction (for example, $5/100$), or as a decimal (for example, 0.05).

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the value is calculated at maturity for a security that has an annual coupon rate 5.5% and a yield of 3.5%. The result is written to the log.

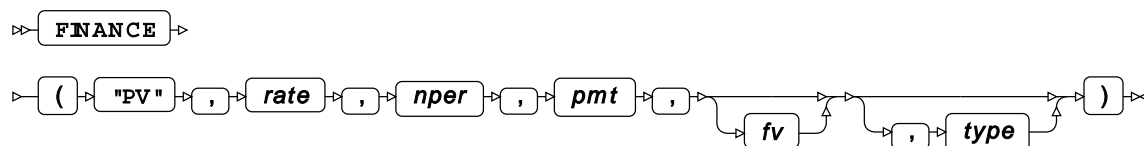
```
DATA _NULL_;
  oddp = FINANCE("PRICEMAT", "01-Feb-2015"d, "31-Mar-2018"d, "01-Jan-2015"d,
    5.5/100, 3.5/100);
  PUT 'The discount price is: ' oddp nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The discount price is: £105.66
```

PV Calculation

Returns the present value of an investment, based on a series of future payments.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

nper

Type: Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

pmt

Type: Numeric

The payment per period.

fv

Optional argument

Type: Numeric

The value of the investment after the specified number of payments.

If this argument is not specified, the default value is 0 (zero).

type

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example – present value of future savings

In this example, the present value is calculated for future savings of £7,200 paid at £120 per month, over five years with an annual interest rate of 2%. The result is written to the log.

```
DATA _NULL_;  
  pv = FINANCE("PV", 2/100, 5, -100);  
  PUT 'The present value is: ' pv nlmnlgbp12.2;  
RUN;
```

pmt has been entered as a negative number as it is an outgoing payment.

The example produces the following output:

```
The present value is: £6,846.28
```

That is, £7,200 in five years at 2% interest a year has a present value of £6,846.28.

Example – present value required for an annuity

In this example, the function calculates the principal – that is, the present value – required if you want to receive an annuity of £500 per month, over 20 years with an annual interest rate of 8%. Interest is applied at the beginning of the period. As the annual interest rate is applied monthly, the `EFFRATE` function has been used to calculate the effective interest rate. The result is written to the log.

```
DATA _NULL_;  
  pv = FINANCE("PV", EFFRATE("MONTH", 8)/12, 20*12, 500,,1);  
  PUT 'The present value is: ' pv nlmnlgbp15.2;  
RUN;
```

pmt has been entered as a positive number as it is an incoming payment.

The example produces the following output:

```
The present value is: (£58,870.00)
```

That is, to ensure the specified monthly payment, you would have to invest £58,870.00. As this is an investment, the value returned is a negative number (shown in parentheses in this format).

Example – present value required for annuity with a retained future value

In this example, the function calculates the principal – that is, the present value – required if you want to receive an annuity of £500 per month, over 20 years with an annual interest rate of 8%, but want to retain £60,000 of the future value. As the annual interest rate is applied monthly, the `EFFRATE` function has been used to calculate the effective interest rate. The result is written to the log.

```
DATA _NULL_;  
  pv = FINANCE("PV", (8/100)/12, 20*12, 500, 60000);  
  PUT 'The present value is: ' pv nlmnlgbp15.2;  
RUN;
```

pmt has been entered as a positive number as it is an incoming payment.

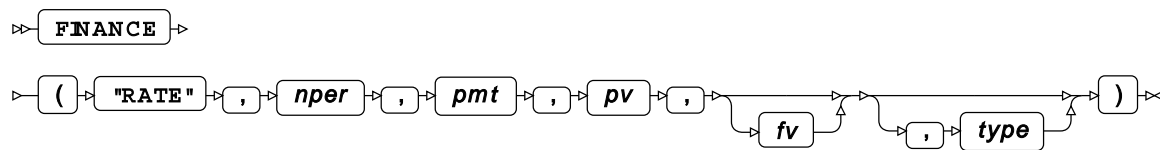
The example produces the following output:

```
The present value is: (£69,939.44)
```

That is, to ensure the specified monthly payment and a future value of £60,000 you would have to invest £69,939.44. As this is an investment, the value returned is a negative number (shown in parentheses in this format).

RATE Calculation

Returns the interest rate required to pay off a specified amount of a loan, or to reach a target amount on an investment, over a number of periods with known payments.



Return type: Numeric

nper

Type: Numeric

The number of periods over which payments will be made to the investment. For example, five years might be specified as 5, or as 60, for 60 months.

pmt

Type: Numeric

The payment amount per period.

As this is an outgoing, this should be entered as a negative number. If you do not enter a negative number, an error message is returned.

pv

Type: Numeric

The current value of the loan or investment.

fv

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Example – interest rate required for specified loan payments

In this example, the interest rate is calculated that would apply to a loan of £6,000 that is fully paid off over 60 months (five years) and paid at £200 per month. The result is written to the log.

```
DATA _NULL_;  
  ir = FINANCE("RATE",60, -200, 6000);  
  PUT 'The interest rate is: ' ir percent7.2;  
RUN;
```

pmt has been entered as a negative number as it is an outgoing payment.

The example produces the following output:

```
The interest rate is: 2.63%
```

Example – interest rate required for specified investment

In this example, the interest rate is calculated to reach a final investment value of £6,000, where £90 is paid monthly over 60 months (5 years). The result is written to the log.

```
DATA _NULL_;  
  ir = FINANCE("RATE",60,-90,0,6000);  
  PUT 'The interest rate is: ' ir percent7.2 ;  
RUN;
```

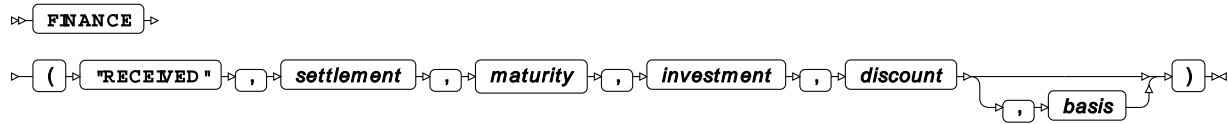
pmt has been entered as a negative number as it is an outgoing payment.

The example produces the following output:

```
The interest rate is: 0.35%
```

RECEIVED Calculation

Returns the amount that will be received at the maturity of a fully-invested security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

investment

Type: Numeric

The value of the investment at maturity.

discount

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Example

In this example, the amount received at maturity is calculated for a security with an initial price of £5,000 that is invested for four years with a discount rate of 3.5%. The result is written to the log.

```
DATA _NULL_;  
  oddp = FINANCE("RECEIVED", "01-Feb-2011"d, "31-Mar-2015"d, 5000, 3.5/100);  
  PUT 'The return is: ' oddp nlmnlgbp12.2;  
RUN;
```

This produces the following output:

```
The return is: £5,853.66
```

SLN Calculation

Returns the straight-line depreciation per period for an asset over a specified period.

➤ **FINANCE** ➤ ("SLN" , *cost* , *salvage* , *life*) ➤

Return type: Numeric

cost

Type: Numeric

The value of the asset before depreciation.

salvage

Type: Numeric

The value of the asset at the end of the specified number of periods.

life

Type: Numeric

The useful life of the asset, specified as a number of periods.

Example

In this example, the straight-line depreciation is calculated for an asset with a life of five years, and initial and salvage costs of £100,000 and £60,000, respectively. The result is written to the log.

```
DATA _NULL_;  
  dac = FINANCE("SLN", 100000, 60000, 5);  
  put 'The depreciation is ' dac nlmnlgbp12.2 'per year';  
RUN;
```

This produces the following output:

```
The depreciation is £8,000.00 per year
```

Note:

Although this example has been described as applying to years, the result would be the same if the periods were months, or quarters, and so on.

SYD Calculation

Returns the depreciation of an asset, using the sum of the years' digits method.

➤ **FINANCE** ➤ ("SYD " , *cost* , *salvage* , *life* , *period*) ➤

The depreciation is calculated for a specified number of years beginning from the first year.

Return type: Numeric

cost

Type: Numeric

The value of the asset before depreciation.

salvage

Type: Numeric

The value of the asset after depreciation.

life

Type: Numeric

The number of periods over which the asset is depreciated.

period

Type: Numeric

The period within the asset's lifetime for which you want the depreciation. For example, if you want the depreciation in the third year of a five-year lifetime, you would specify 3.

Example

In this example, the value of the depreciation is calculated for year four of a seven-year depreciation on an asset worth £173,500. The result is written to the log.

```
DATA _NULL_;  
  P = 4;  
  diy = FINANCE("SYD", 173500, 20000, 7, p);  
  PUT 'The depreciation for year ' p 'is:' diy 10.2;  
RUN;
```

This produces the following output:

```
The depreciation for year 4 is: 21928.57
```

TBILLEQ Calculation

Returns the bond-equivalent yield for a Treasury Bill based on a specified discount rate.

⇒ **FINANCE** ⇒ ("TBILLEQ" , *settlement* , *maturity* , *discount*) ⇒

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the bill will be repaid. This must be later than *settlement*, otherwise an error message is returned, but should also be within one year of *settlement*.

discount

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield is calculated for a treasury bill held for eleven months with a discount rate of 5%. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("tbilleq", mdy(07,11,2014), mdy(07,10,2015),5/100);  
  PUT 'The yield is: ' ret percent7.2;  
RUN;
```

This produces the following output:

```
The yield is: 5.27%
```

TBILLPRICE Calculation

Returns the price, per \$100 face value, of a Treasury Bill.

➤ **FNANCE** ➤ ("TBILLPRICE" , *settlement* , *maturity* , *discount*) ➤

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the bill will be repaid. This must be later than *settlement*, otherwise an error message is returned, but should also be within one year of *settlement*.

discount

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the price is calculated for a treasury bill held for eleven months with a discount rate of 5%. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("tbillprice", mdy(07,11,2014), mdy(07,10,2015),5/100);  
  PUT 'The price is: ' ret nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The price is: £94.94
```

TBILLYIELD Calculation

Returns the percentage yield of a Treasury Bill with a \$100 face value.

➤ **FINANCE** ➤ ("TBILLYIELD" , *settlement* , *maturity* , *pr*) ➤

Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the bill will be repaid. This must be later than *settlement*, otherwise an error message is returned, but should also be within one year of *settlement*.

pr

Type: Numeric

The price of the treasury bill. This is the price relative to the \$100 face price that will be used to calculate the yield.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the percentage yield is calculated for a treasury bill held for eleven months with a purchase price of £95. The result is written to the log.

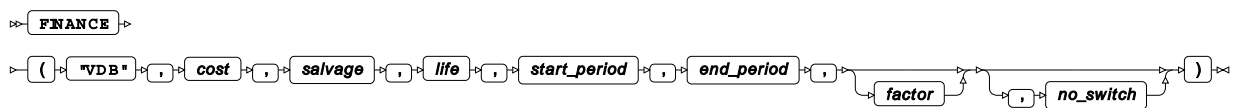
```
DATA _NULL_;
  ret = FINANCE("tbillyield", mdy(07,11,2014), mdy(07,10,2015),95);
  PUT 'The yield is: ' ret percent7.2;
RUN;
```

This produces the following output:

```
The yield is: 5.21%
```

VDB Calculation

Returns the depreciation of an asset. By default, the function uses the double declining balance method, but you can specify a different depreciation rate if required.



The depreciation is calculated over specified periods.

Return type: Numeric

cost

Type: Numeric

The value of the asset before depreciation.

salvage

Type: Numeric

The value of the asset at the end of the specified number of periods.

life

Type: Numeric

The useful life of the asset, specified as a number of periods.

start_period

Type: Numeric

The period from which depreciation is calculated.

end_period

Type: Numeric

The last period for which depreciation is calculated.

factor

Optional argument

Type: Numeric

The depreciation rate. Typically, 2 (for 200%), would be used to specify a double-declining balance.

no_switch

Optional argument

Type: Numeric

Specifies whether the function should calculate depreciation using a straight line method if the straight line depreciation would be greater than that calculated using the declining balance method:

1

Do not switch to the straight-line depreciation method. This is the default.

0

Switch to the straight-line depreciation method.

Basic example

In this example, the depreciation is calculated for years two through four of an asset with an initial value of £100,000, a salvage value of £25,000 and a lifetime of five years. The default double declining balance method is used. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("VDB", 100000, 25000,5,2,4);  
  PUT 'The amount is: ' ret nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The amount is: £23,040.00
```

Example – depreciation with specified rate

In this example, the depreciation is calculated for years two through four of an asset with an initial value of £100,000, a salvage value of £25,000, a lifetime of five years, and a depreciation rate of 1.75%. The result is written to the log.

```
DATA _NULL_;  
  ret = FINANCE("VDB", 100000, 5000,5,2,4,1.75);  
  PUT 'The amount is: ' ret nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The amount is: £24,399.38
```

Example – converting to straight line depreciation

In this example, the depreciation is calculated for years two through four of an asset with an initial value of £100,000, a salvage value of £25,000, a lifetime of five years, and a depreciation rate of 1.75%. The switch is set so that the straight line depreciation method will be used if necessary. The result is written to the log.

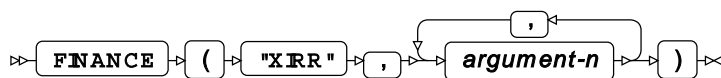
```
DATA _NULL_;
  ret = FINANCE("VDB", 100000, 5000,5,2,4,1.5,0);
  PUT 'The amount is: ' ret nlnmlgbp12.0;
RUN;
```

This produces the following output:

```
The amount is: £29,350.00
```

XIRR Calculation

Returns the internal rate of return for specified cash flows.



The cash flows are specified as a set of values that includes an initial investment value and a series of net incomes, and the dates on which those payments are made.

Unlike the FINANCE IRR [↗](#) or IRR [↗](#) (page 1555) functions, the series of cashflows specified in this function do not have to be periodic.

Return type: Numeric

argument-n

Type: Numeric

This argument consists of a pair of values, a date and a payment made at that date. The date can be entered in any of the ways in which dates can be represented in WPS, such as a numeric value or using a function. For example, if you want to enter a payment of £1,000, received on the 10th of August 2016, you could specify this argument as:

```
mdy(08,10,2016), 1000
```

The values entered for *argument-n* represent the payments made to the investment and the payments received. The first pair of values usually represent the payment and the date it was made. A payment should be entered as a negative number. See the section below for an example of specifying values for *argument-n*.

Example

In this example, the internal rate of return is calculated for an investment of £10,000 with subsequent cash flows received on the specified dates. The result is written to the log.

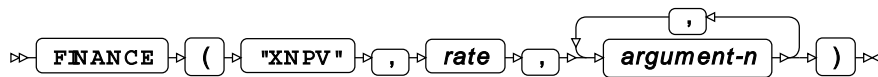
```
DATA _NULL_;
  ret = FINANCE("XIRR", mdy(01,01,2016),-10000, mdy(02,01,2016), 2000,
    mdy(05,01,2016),2400, mdy(07,01,2016), 2900, mdy(11,01,2016), 3500,
    mdy(01,01,2017),4100);
  PUT 'The internal rate of return is: ' ret percent7.2;
RUN;
```

This produces the following output:

```
The internal rate of return is: 4.84%
```

XNPV Calculation

Returns the Net Present Value for a series of cash flows; these cash flows do not have to be periodic.



Return type: Numeric

rate

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

argument-n

Type: Numeric

Either a cash flow, or the date a cash flow was received.

Entries need to be paired; a cash flow must correspond to its received date. There would typically be a series of cash flows and dates. The function expects cash flows to be grouped together first, and then the dates corresponding to these cash flows. For example:

```
-10000, 2000, 2400, 2900, "01-Jan-2016"d, "01-Feb-2016"d, "01-May-2016"d, "01-Jul-2016"d
```

A cash flow might be a payment to an investment, or a payment received. A payment to an investment must be entered as a negative number.

Dates can be entered in any of the ways in which they can be represented in WPS, such as a numeric value, a function or a date literal. For example, if you want to enter a payment of £1,000, received on the 10th of August 2016, you could specify this argument as:

```
mdy(08,10,2016), 1000
```

If the number of dates and cash flows do not match, an error message is returned.

Example

In this example, the net present value is calculated for an investment of £10,000 with subsequent cash flows received on the specified dates. The discount rate is 5%. The result is written to the log.

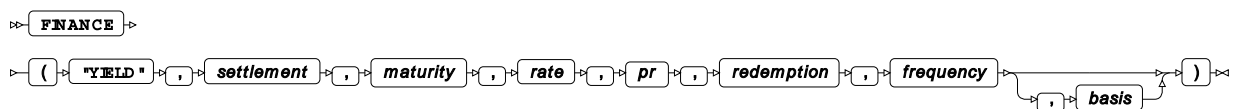
```
DATA _NULL_;
  ret = FINANCE("XNPV", 5/100, mdy(01,01,2016),-10000, mdy(02,01,2016), 2000,
    mdy(05,01,2016),2400, mdy(07,01,2016),2900,mdy(11,01,2016),3500,
    mdy(01,01,2017),4100);
  PUT 'The net present value: ' ret nlnmlgbp12.2;
RUN;
```

This produces the following output:

```
The net present value: (£4,731.52)
```

YIELD Calculation

Returns the yield for a security that pays periodic interest for specified periodic cash flows.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The annual coupon rate.

pr

Type: Numeric

The price of the security. The face value is assumed to be \$100.

redemption

Type: Numeric

The redemption value of the security. The face value is assumed to be \$100.

frequency

Type: Numeric

The number of coupon payments per year; this must be 1, 2 or 4. If you use any other value, an error message is returned.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield is calculated for a bond with a par value of £100 bought on January 10th 2016 for a price of £97. The coupon rate is 5%, and coupon payments are made twice a year for the next two years. The redemption value is expected to be £102. The result is written to the log.

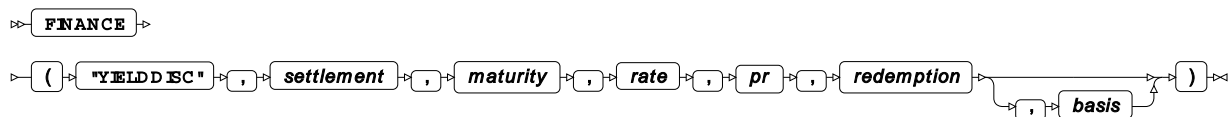
```
DATA _NULL_;
  yp= FINANCE("yield", "10-Jan-2016"d, "10-Jan-2018"d, (5/100), 97, 102, 2);
  PUT 'The yield is: ' yp percent10.2;
run;
```

This produces the following output:

```
The yield is:      7.59%
```

YIELDDISC Calculation

Returns the annual yield of a discounted security.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The annual coupon rate.

pr

Type: Numeric

The price of the security. The face value is assumed to be \$100.

redemption

Type: Numeric

The redemption value of the security. The face value is assumed to be \$100.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield of a discounted security is calculated for a bond bought on January 10th 2016 for a price of £97. The redemption value is expected to be £102. The result is written to the log.

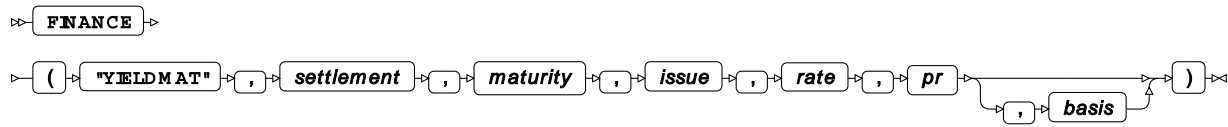
```
DATA _NULL_;  
  yp= FINANCE("YIELDDISC","10-Jan-2016"d, "10-Jan-2018"d,97,102,3);  
  PUT 'The yield to maturity is: ' yp percent10.2;  
RUN;
```

This produces the following output:

```
The yield to maturity is:      2.57%
```

YIELDMAT Calculation

Returns the annual yield of a security that pays interest at maturity.



Return type: Numeric

settlement

Type: Numeric

The settlement date of the security. This must be earlier than *maturity*, otherwise an error message is returned.

maturity

Type: Numeric

Date on which the security matures; that is, the date on which the coupon expires. This must be later than *settlement*, otherwise an error message is returned.

issue

Type: Numeric

Date on which the security was issued. This must be earlier than *settlement*, otherwise an error message is returned.

rate

Type: Numeric

The annual coupon rate.

pr

Type: Numeric

The price of the security. The face value is assumed to be \$100.

basis

Optional argument

Type: Numeric

The day count basis used by the function for the calculation. This can be one of the following:

0

30/360 (US). This is the default. A month is treated as having 30 days, the year as 360. Adjustments use US conventions.

1

Actual/actual. Months and years are treated as having their actual number of days.

2

Actual/360. A month is treated as having the actual number of days, the year as 360.

3

Actual/365. A month is treated as having the actual number of days, the year as 365.

4

30/360 (European). A month is treated as having 30 days, the year as 360. Adjustments use European conventions.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

Example

In this example, the yield to maturity is calculated for a bond with a par value of £100 bought on January 10th 2016 for a price of £97. The result is written to the log.

```
DATA _NULL_;
  yp= FINANCE("yieldmat","10-Jan-2016"d, "10-Jan-2018"d,"10-Dec-2015"d, 5/100, 97);
  PUT 'The yield to maturity is: ' yp percent10.2;
RUN;
```

This produces the following output:

```
The yield to maturity is:      6.67%
```

GARKHCLPRC

Returns the price for a European call option using the Garman-Kohlhagen model.

```
➤ GARKHCLPRC ➤ ( exercise-price , time , spot-price , rate-domestic , rate-foreign , volatility ) ➤
```

Return type: Numeric

exercise-price

Type: Numeric

The exercise or current price for the call option.

time**Type:** Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

spot-price**Type:** Numeric

The assumed strike price for the option.

rate-domestic**Type:** Numeric

The domestic risk free simple interest rate over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100

rate-foreign**Type:** Numeric

The foreign risk free simple interest rate over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100

volatility**Type:** Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European call option that will expire in four months. The exercise price and future price are both €20; the volatility is 25% per annum, the domestic risk-free rate is 9% per annum, and the foreign risk-free rate is 15% per annum. The result is written to the log.

```
DATA _NULL_;  
  cp=garkhclprc(20, 0.333, 20, 0.09, 0.15, 0.25);  
  PUT 'The call price is: ' cp euro5.2;  
RUN;
```

This produces the following output:

```
The call price is: E0.92
```

You could also enter *time* as a fraction:

```
cp=garkhclprc(20, 4/12, 20, 0.09, 0.15, 0.25);
```

This would return the same result.

GARKHPTPRC

Returns the price for a European put option using the Garman-Kohlhagen model.

➤ **GARKHPTPRC** ➤ (*exercise-price* , *time* , *spot-price* , *rate-domestic* , *rate-foreign* , *volatility*) ➤

Return type: Numeric

exercise-price

Type: Numeric

The exercise or current price for the call option.

time

Type: Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

spot-price

Type: Numeric

The assumed strike price for the option.

rate-domestic

Type: Numeric

The domestic risk free simple interest rate over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100

rate-foreign

Type: Numeric

The foreign risk free simple interest rate over the life of the option. For example, if the rate is 9%, you can enter 0.09, or 9/100

volatility

Type: Numeric

The annualised future price volatility; this should be a positive decimal number. For example, if the volatility is 25%, you can enter 0.25, or 25/100.

Example

In this example, the function returns the price of a European put option that will expire in four months. The exercise price and future price are both €20; the volatility is 25% per annum, the domestic risk-free rate is 9% per annum, and the foreign risk-free rate is 15% per annum. The result is written to the log.

```
DATA _NULL_;
  cp=garkhclprc(20, 0.333, 20, 0.09, 0.15, 0.25);
  PUT 'The call price is: ' cp euro5.2;
RUN;
```

This produces the following output:

```
The put price is: E1.31
```

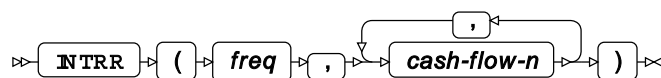
You could also enter *time* as a fraction:

```
pp=garkhptprc(20, 4/12, 20, 0.09, 0.15, 0.25);
```

This would return the same result.

INTRR

Returns the internal rate of return for a supplied series of periodic cashflows.



Return type: Numeric

The value is returned as a decimal number which can be formatted as a percentage using the `PERCENT` format or multiplied by 100 to generate a percentage.

freq

Type: Numeric

The number of payments made per period. For example, if you specify 2, then two payments are made per period, two per month or two per year. Specify 0 (zero) to enable continuous compounding.

cash-flow-n

Type: Numeric

A cashflow for a period.

The first cashflow specified should be the initial investment, and should thus be entered as a negative number.

Example

In this example, the internal rate of return is calculated for a supplied series of cashflows. The frequency is specified as 2, meaning there are two payments a period. There are six payments, so the cashflows are spread over three years. The result is written to the log.

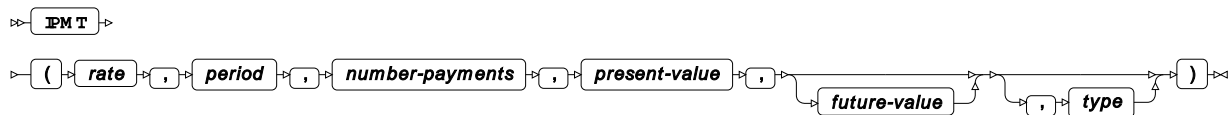
```
DATA _NULL_;
  rr = intrr(2, -100, 20, 24, 28.80, 34.56, 41.47, 20);
  PUT 'The internal rate of return is: ' rr percent10.4;
RUN;
```

This produces the following output:

```
The internal rate of return is: 35.0225%
```

IPMT

Returns the interest paid for a specific period of a loan or investment that is being paid with constant periodic payments and has a constant interest rate.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

period

Type: Numeric

The period for which the interest payment is calculated. This must be an integer between 1 and *number-payments*. For example, this might be two years, in which case you would specify 2, or 24 months, in which case you would specify 24.

number-payments**Type:** Numeric

The number of payments required to repay the loan or pay the investment. For example, this might be five years, in which case you would specify 5, or 60 months, in which case you would specify 60.

present-value**Type:** Numeric

The current value of the loan or investment.

future-value

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

Example

In this example, the interest paid is calculated for a loan that has an interest rate of 3.45% and a present value of £5,000. The interest to be paid is calculated for each year of a five year loan period. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;
  t = 0;
  DO i = 1 TO 5;
    ip= IPMT((3.45/100), i, 5, 5000, 0);
    PUT 'The interest for the period (year) ' i 'is: ' ip 10.2;
    t = t + ip;
  END;
  PUT 'Total interest paid: ' t 10.2;

RUN;
```


This produces the following output:

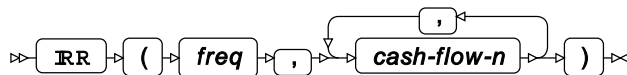
The interest for the period (year) 1 is: :	172.50
The interest for the period (year) 2 is: :	140.30
The interest for the period (year) 3 is: :	106.99
The interest for the period (year) 4 is: :	72.53
The interest for the period (year) 5 is: :	36.88
Total interest paid:	529.20

Note:

This result is the same as the result of the `CUMIPMT` function for five years. See the first example in the section describing `CUMIPMT` [\(page 1447\)](#) function.

IRR

Returns the internal rate of return for a supplied series of periodic cashflows.



Return type: Numeric

The value is returned as a decimal number which can be formatted as a percentage using the `PERCENT` format or multiplied by 100 to generate a percentage.

freq

Type: Numeric

The number of cashflows.

cash-flow-n

Type: Numeric

A cashflow for a period.

The first cashflow specified should be the initial investment, and should thus be entered as a negative number.

If the cash flows are non-periodic, you should use the `FINANCE XIRR` [\(page 1542\)](#) function.

Example

In this example, the internal rate of return is calculated for a supplied series of cashflows. The result is written to the log.

```
DATA _NULL_;  
  rr = irr(1, -100, 20, 24, 28.80, 34.56, 41.47);  
  PUT 'The internal rate of return is: ' rr;  
RUN;
```

This produces the following output:

```
The internal rate of return is: 13.1%
```

MARGRCLPRC

Returns the price for a European call option using Margrabe's formula.

⇒ **MARGRCLPRC** ⇒ (*s1* , *time* , *s2* , *sigma1* , *sigma2* , *rho*) ⇒

Return type: Numeric

s1

Type: Numeric

The price of the first asset.

time

Type: Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

s2

Type: Numeric

The price of the second asset.

sigma1

Type: Numeric

The volatility of *s1*.

sigma2

Type: Numeric

The volatility of *s1*.

rho**Type:** NumericThe correlation between *s1* and *s2*.**Example**

In this example, the function returns the price of a European call option that will expire in four months. The asset prices are both €20; the volatility each asset is 25% and 30% per annum, and the correlation between the two asset prices is 20%. The result is written to the log.

```
DATA _NULL_;  
  cp=margrclprc(20, 0.333, 20, 0.25, 0.3, 0.15);  
  PUT 'The call price is: ' cp euro5.2;  
RUN;
```

This produces the following output:

```
The call price is: E1.66
```

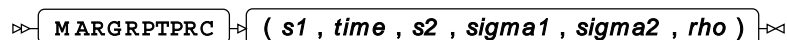
You could also enter *time* as a fraction:

```
cp=margrclprc(20, 4/12, 20, 0.25, 0.3, 0.15);
```

This would return the same result.

MARGRPTPRC

Returns the price for a European put option using Margrabe's formula.



```
MARGRPTPRC ( s1 , time , s2 , sigma1 , sigma2 , rho )
```

Return type: Numeric***s1*****Type:** Numeric

The price of the first asset.

time**Type:** Numeric

The number of years until the option expires. Must be greater than 0. For example, six months could be specified as 6/12 or as 0.5; two years could be entered as 2 or 24/12.

s2**Type:** Numeric

The price of the second asset.

sigma1

Type: Numeric

The volatility of *s1*.

sigma2

Type: Numeric

The volatility of *s2*.

rho

Type: Numeric

The correlation between *s1* and *s2*.

Example

In this example, the function returns the price of a European put option that will expire in four months. The asset prices are both €20; the volatility is 25% and 30% per annum, and the correlation between the two asset prices is 20%. The result is written to the log.

```
DATA _NULL_;  
  pp=margrptprc(20, 0.333, 20, 0.25, 0.3, 0.15);  
  PUT 'The put price is: ' pp euro5.2;  
RUN;
```

This produces the following output:

```
The put price is: E1.66
```

You could also enter *time* as a fraction:

```
pp=margrptprc(20, 4/12, 20, 0.25, 0.3, 0.15);
```

This would return the same result.

MORT

Returns information about the amortisation of a loan.

➤ **MORT** ➤ **(*p* , *a* , *i* , *n*)** ➤

You can calculate one of:

- The amount of a loan (*p*)
- The payment at each period required to amortise the loan (*a*)

- The interest rate of the loan (i)
- The period over which the loan will be amortised (n)

if you provide to the function the corresponding information about the loan. That is, you can calculate the interest rate if you know the amount of the loan, the number of payments and the amount to be paid at each period. Similarly, you can calculate the size of the loan if you know the interest rate, the number of payments and the amount to be paid at each period. You do this by omitting the relevant argument and supplying the others. The section *Example* provides example of usage.

Return type: Numeric

p

Type: Numeric

The loan principal.

a

Type: Numeric

The repayment amount.

i

Type: Numeric

The interest rate.

n

Type: Numeric

The number of payments

The omitted arguments can be represented by nulls, or by missing values:

```
mort(50000,, 5/100, 60)
```

is equivalent to:

```
mort(50000,., 5/100, 60)
```

Example

In this example, the function first returns the repayment amount for a loan of £50,000 over five years with an interest rate of 5%. Having obtained the repayment, the function is then used to calculate values for the other arguments. The [CUMIPMT](#) (page 1447) function is also used to get the cumulative interest so that the total amount repaid can be shown. The result is written to the log.

```
DATA _NULL_;

am=mort(50000,.,0.05,5);
PUT 'The yearly repayments are: ' am nlmnlgbp12.2;

am=mort(50000,11548.74,.,5);
PUT 'The interest rate is: ' am percent6.3;

am=mort(50000, 11548.74 ,0.05,);
PUT 'The number of repayments is: ' am 3.;

am=mort(,11548.74 ,0.05,5);
PUT 'The amount to be repaid is: ' am nlmnlgbp12.0;

totwint = cumipmt(0.05,5,50000) + 50000;

PUT 'Total repaid including cumulative interest: ' totwint nlmnlgbp12.2;

RUN;
```

This produces the following output:

```
The yearly repayments are: £11,548.74
The interest rate is: 5.0%
The number of repayments is: 5
The amount to be repaid is: £50,000.00
Total repaid including cumulative interest: £57,743.70
```

NETPV

Returns the net present value of an investment, based on a specified discount rate.



Return type: Numeric

rate

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

freq**Type:** Numeric

The number of payments made per period. For example, if you specify 2, then two payments are made per period, two per month or two per year. Specify 0 (zero) to enable continuous compounding.

c0**Type:** Numeric

The initial investment. This should be entered as a negative number.

c-n**Type:** Numeric

The payment per period.

A period is unit-less, and is not specified in the function. The period is whatever is of interest to you. For example, if you enter a value for *freq* of 1, the rate is applied once to the period; that will therefore be yearly, if that is the period of interest, or monthly if that is the period of interest. Similarly, a value of 2 is applied twice to the period, which will be twice a year, twice a month, or twice a day, and so on; while a value of 0.5 would apply the rate every other period every other year or month, and so on. The cash flows are made at each subsequent period to the initial investment. These cash flows follow the same *freq*; if *freq* is 1, the payments are once a period, if 2, twice a period, if 0.5, every other period, and so on. A value of 0.5 would apply the rate every other year, or every other month, and so on.

Basic example

In this example, the net present value is calculated for an investment of £5,000 with the subsequent specified cash flows. The discount rate is 2%, and the frequency of payment is 2 (that is, twice a year, or twice a month, and so on). The result is written to the log.

```
DATA _NULL_;  
  npv=netpv(2/100,2,-5000, 800, 950, 1080, 1220, 1500, 300);  
  PUT 'The net present value is: ' npv nlmnlgbp12.2;  
RUN;
```

In this example, the period is 2, so payments and discounts occur twice each period. As there are six cash flows, the NPV is calculated over three years, if the period of interest is yearly, or over six months, if the period of interest is monthly. The example produces the following output:

This produces the following output:

```
The net present value is: £654.75
```

Example – four payments a period

In this example, the net present value is calculated for an investment of £5,000 with subsequent specified cash flows and discounted four times a period. The discount rate is 2%. The result is written to the log and to the example dataset.

```
DATA _NULL_;
  npv=netpv(2/100, 4, -5000, 800, 950, 1080,1220, 1500, 950,
           1000, 1080, 1220, 1500, 950, 800);
  PUT 'The net present value is: ' npv nlmnlgbp12.2;
RUN;
```

In this example, the period is four. Thus, if the period is a year, there would be four payments a year. The example produces the following output:

```
The net present value is: £7,634.26
```

Example – payments every other period

In this example, the net present value is calculated for an investment of £5,000 with subsequent specified cash flows and discounted every other period. The discount rate is 2%. The result is written to the log.

```
DATA _NULL_;
  npv=netpv(2/100, 1/2, -5000, 800, 950, 1080, 1220, 1500, 950,
           1000, 1080, 1220, 1500, 950, 800);
  PUT 'The net present value is: ' npv nlmnlgbp12.2;
RUN;
```

In this example, the period is set to 0.5, which implies a discount and return every other period. Thus, if the period is a year, the rate would be applied every other year; if the period is a month, the rate is applied every two months, and so on. Payments would also be made at the same frequency, so if the period of interest is a year, a payment would be made and a discount applied every other year; if the period is a month, a payment would be made and a discount applied every other month. The example produces the following output:

```
The net present value is: £5,146.57
```

NOMRATE

Returns the nominal annual interest rate.

➤ **NOMRATE** ➤ (*period* , *rate*) ➤

The period over which the nominal rate is calculated is defined as a year. You can define the periodicity of compounding over the year.

Return type: Numeric

period

The compounding period.

rate

Type: Numeric

The number of compounding periods per year.

Example

In this example, the nominal interest rate is returned for an interest rate of 2.5% per annum compounded quarterly. The result is written to the log.

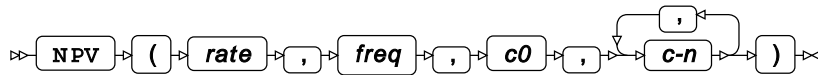
```
DATA _NULL_;  
nr=NOMRATE('quarter',2.5);  
PUT 'The nominal interest rate is: ' nr percent7.2;  
RUN;
```

This produces the following output:

```
The nominal interest rate is: 2.48%
```

NPV

Returns the net present value of an investment, based on a specified discount rate. The rate is specified as a percentage.



Return type: Numeric

rate

Type: Numeric

The discount rate. The rate must be specified as a fraction or decimal number. For example, 5% should be entered as 5/100 or as 0.05.

freq

Type: Numeric

The frequency at which the rate is applied for a period.

c0

Type: Numeric

The initial investment. This should be entered as a negative number.

c-n

Type: Numeric

The payment per period.

A period is unit-less, and is not specified in the function. The period is whatever is of interest to you. For example, if you enter a value for *freq* of 1, the rate is applied once to the period; that will therefore be yearly, if that is the period of interest, or monthly if that is the period of interest. Similarly, a value of 2 is applied twice to the period, which will be twice a year, twice a month, or twice a day, and so on; while a value of 0.5 would apply the rate every other period every other year or month, and so on. The cash flows are made at each subsequent period to the initial investment. These cash flows follow the same *freq*; if *freq* is 1, the payments are once a period, if 2, twice a period, if 0.5, every other period, and so on. For example, a value of 0.5 would apply the rate every other year, or every other month, and so on.

Basic example

In this example, the net present value is calculated for an investment of £5,000 with the subsequent specified cash flows. The discount rate is 2%, and the frequency of payment is 1. The result is written to the log.

```
DATA _NULL_;
  pv=npv(2/100,1,-5000, 800,950,1080,1220,1500);
  PUT 'The net present value is: ' pv nlmnlgbp12.2;
RUN;
```

In this example, the period is 1, which implies payments and discounting every year if the period is yearly. As there are five cash flows, the NPV is calculated over five years if the period of interest is yearly.

This produces the following output:

```
The net present value is: £546.34
```

Example – four payments per period

In this example, the net present value is calculated for an investment of £5,000 with subsequent specified cash flows and discounted four times a year. The discount rate is 2%. The result is written to the log.

```
DATA _NULL_;
  npv=netpv(2, 4, -5000, 800, 950, 1080,1220, 1500, 950,
            1000, 1080, 1220, 1500, 950, 800);
  PUT 'The net present value is: ' npv nlmnlgbp12.2;
RUN;
```

The period is four, which creates four discounts. Thus, if the period is a year, the rate would be applied four times in the year; if the period is monthly, the rate would be applied approximately every week, and so on. Payments would also be made at the same frequency; if the period of interest is a year, four payments a year would be made over three years, while if the period is a month, then four payments a month would be made over three months.

This produces the following output:

```
The net present value is: £7,634.26
```

Example – payments every other period

In this example, the net present value is calculated for an investment of £5,000 with subsequent specified cash flows and discounted every other period. The discount rate is 2%. The result is written to the log.

```
DATA _NULL_;
  npv=npv(2, 1/2, -5000, 800, 950, 1080, 1220, 1500, 950,
          1000, 1080, 1220, 1500, 950, 800);
  PUT 'The net present value is: ' npv nlmnlgbp12.2;
RUN;
```

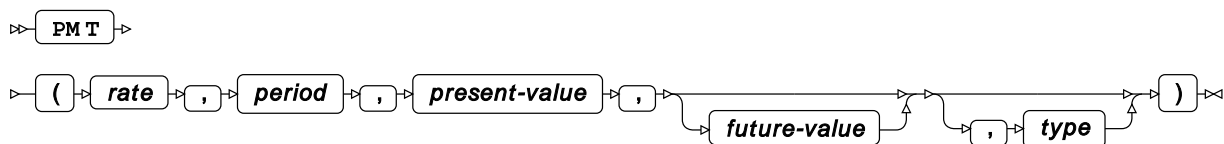
The period is set to 0.5, which implies a discount and return every other period. Thus, if the period is a year, the rate would be applied every other year; if the period is a month, the rate is applied every two months, and so on. Payments would also be made at the same frequency; so, if the period of interest is a year, a payment would be made and a discount applied every other year; if the period is a month, a payment would be made and a discount applied every other month.

This produces the following output:

```
The net present value is: £5,146.57
```

PMT

Returns the periodic payment necessary to pay off a loan, where the interest rate is constant.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

period

Type: Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

present-value

Type: Numeric

The current value of the loan or investment.

future-value

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If this argument is not specified, the default value is 0 (zero).

You can omit *future-value*, but if you do so and want to specify *type*, you should ensure you demarcate the missing option with commas; for example:

```
pmt(0.05, 5, 50000,,0)
```

Basic example

The following example calculates the monthly payment required to completely repay a loan over five years. The payments are made monthly. The interest is calculated at the end of each period. The `EFFRATE` function has been used to calculate the effective interest rate at each period. The result is written to the log.

```
DATA _NULL_;
  pymt=pmt(effrate("month", 0.05/12), 60, 50000);
  PUT 'The payments are: ' pymt nlmnlgbp12.2;
RUN;
```

This produces the following output:

```
The payments are: £834.39
```

Paying for an investment

The following example calculates the quarterly payments for a five year investment with a current value of £20,000 and a future value of £50,000. The interest is calculated at the end of each period. The `EFFRATE` function has been used to calculate the effective interest rate at each period. The result is written to the log and to the example dataset.

```
DATA _NULL_;
  pymt=pmt(effrate("quarter", 0.05/12), 5*4, 20000, 50000,1);
  PUT 'The payments are: ' pymt nlmnlgbp12.2;
RUN;
```

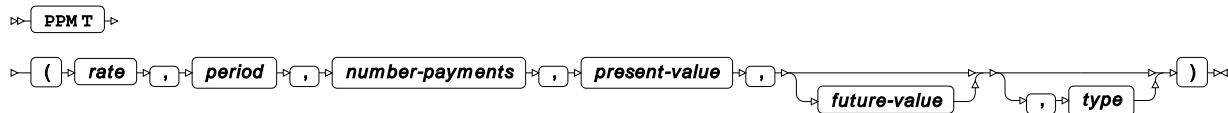
This produces the following output:

```
The payments are: £3,499.30
```

The loan period of five years is multiplied by four to specify the correct number of repayment periods.

PPMT

Returns the payment necessary to repay the principal of a loan or to pay into an investment for a specified period, where the interest rate is constant and periodic payments are made.



Return type: Numeric

rate

Type: Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

period

Type: Numeric

number-payments

Type: Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

present-value

Type: Numeric

The current value of the loan or investment.

future-value

Optional argument

Type: Numeric

The value of the loan or investment after the specified number of payments. If this is not specified, the default is 0.

type

Optional argument

Type: Numeric

Specify whether payments are made at the beginning or end of a period. Enter one of the following values:

0

Payment is made at the end of the period. This is the default.

1

Payment is made at the beginning of the period.

If this argument is not specified, the default value is 0 (zero).

Example

In this example, the interest paid is calculated for a loan that has an interest rate of 3.5% and a present value of £5,000. The interest to be paid is calculated for each year of a five year loan period. The interest is calculated at the end of each period. The result is written to the log.

```
DATA _NULL_;  
  i = 3;  
  pymt=pmt(5/100, i, 5, 50000);  
  PUT 'The payment for year ' i 'is : ' pymt nlmnlgbp12.2;  
RUN;
```

This produces the following output:

```
The payment for year 3 is : £9,976.24
```

PVP

Returns the present value of a future amount, where the initial capital is repaid at maturity.

➤ **PVP** ➤ (*par-value* , *rate* , *numeric* , *K* , *k0* , *yield*) ➤

Return type: Numeric

par-value

Type: Numeric

The par value or face value of a bond.

rate

Type: Numeric

The coupon rate of the bond.

numeric

Type: Numeric

Number of coupons per period.

K

Type: Numeric

The number of remaining coupon payments

k0

Type: Numeric

The time to the first coupon. This should be greater than 0 and less than $1/\text{numeric}$. For example, if coupons are monthly, and there are 28 days until the next coupon, you might enter this value as $28/30 * 1/12$, or as 0.078.

yield

Type: Numeric

Yield per period to maturity.

Example

In this example, the present value of an investment is calculated for a coupon that has an interest rate of 3.5%, a par value of £95, coupon payments twice a year with four further payments until expiry, and with a yield of 6.5%. There are two months until the next coupon payment. The result is written to the log.

```
DATA _NULL_;  
  ret = pvp(95, 3.5/100, 1, 5, 11/09/2014, 6.5/100);  
  PUT 'The present value is: ' ret nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The present value is: £89.83
```

SAVING

Returns the amount that would be saved based on a specified interest rate, term and payments.

➤ **SAVING** ➤ (*value* , *payment* , *rate* , *term*) ➤

You can also use this function to calculate the interest rate, term or payments when you have any of the other values. For example, if you know the amount saved, the number of years over which savings occurred, and the payments you can determine the interest rate applied.

Return type: Numeric

value

Type: Numeric

The amount saved.

payment

Type: Numeric

The monthly payment amount.

rate**Type:** Numeric

The interest rate. The rate must be specified as a fraction or decimal number, not as a percent. For example, 5.25% should be entered as 5.25/100 or as 0.0525, not as 5.25 or 5.25%. The rate applies to a period. For an interest rate specified annually, this would be year. The interest rate you specify should take account of the periods used to specify *period*.

If the period specified is monthly, and the rate yearly, you will need to use a function such as `EFFRATE` to first calculate the monthly interest rate. You might need to do the same if the period specified is quarterly, semi-annually, and so on.

term**Type:** Numeric

The number of periods over which payments will be made to the loan or investment. For example, five years might be specified as 5, or as 60, for 60 months.

The argument for which you want a value returned can be represented using a period (.) or a null.

Example – calculating the saved amount

In this example, the function calculates the saved amount after an interest rate of 2% has been applied to monthly payments of £500 a month for five years. The result is written to the log.

```
DATA _NULL_;  
  ret = saving(,500,effrate("month",2)/12,60);  
  PUT 'The amount saved: ' ret nlmnlgbp12.0;  
RUN;
```

This produces the following output:

```
The amount saved: £31,591.23
```

Example – calculating payments

In this example, the function calculates what monthly payments would be required to save £31,576.22 over five years at an interest rate of 2%. The result is written to the log.

```
DATA _NULL_;  
  ret = saving(31576.22,,effrate("month",2)/12,60);  
  PUT 'The amount to save: ' ret nlmnlgbp10.0;  
RUN;
```

This produces the following output:

```
The amount to save: £499.76
```

SAVINGS

Returns the value of savings based on consistent deposits.



The interest rates can be varied over time if required.

Return type: Numeric

base_date

Type: Numeric

The end date on which the calculation is based.

initial_deposit_date

Type: Numeric

The date on which the first deposit is made.

deposit_amount

Type: Numeric

The amount deposited each month.

deposit_number

Type: Numeric

The number of deposits that will be made over the course of the period.

deposit_interval

Type: Character

The interval at which each deposit is made, such as monthly or quarterly. See below for information on the values you can enter.

compounding_interval

Type: Character

The interval at which compounding of interest occurs, such as monthly or quarterly.

An interval can be:

'DAY'

Compounding occurs at the end of each day.

'SEMIMONTH'

Compounding occurs at the middle and end of each month.

'MONTH'

Compounding occurs at the end of each months.

'QUARTER'

Compounding occurs at the end of each quarter year.

'SEMIYEAR'

Compounding occurs at the end of each half year.

'YEAR'

Compounding occurs at the end of each year.

date-1

Type: Numeric

The date at which the interest rate *rate-1* is valid.

rate-1

Type: Numeric

The interest rate expected at *date-1* entered as a percentage, for example 5 for 5%.

date-or-rate-n

Optional argument

Type: Numeric

A pair of comma-separated values. The first is a date at which an interest rate *rate-n* becomes valid, and the second is the interest rate at that date. The date and interest rate must be specified as pairs. If the function finds unpaired values, an error occurs.

Dates are expected in numeric date format; if you want to specify dates in standard forms, you will need to use a suitable date time function or date literal, or format the data on input.

date-1 sets the date from which access to the principal begins. *date-1* cannot therefore be earlier than *initial_deposit_date*.

Example – calculating the saved amount

In this example, the function calculates the amount saved, including interest, where deposits are £500 over 36 months. The result is written to the log.

```
DATA _NULL_;
  ret = savings(mdy(06,11,2017), mdy(07,11,2014), 500, 36, 'month', 'month',
               mdy(07,11,2014), 5);
  PUT 'The amount saved: ' ret nlmnlgbp12.0 ;
RUN;
```

This produces the following output:

```
The amount saved: £18,900.63
```

Example – calculating the saved amount with changes in interest rate

In this example, the function calculates the amount saved, including interest, where deposits are £500 over 36 months. The result is written to the log.

```
DATA _NULL_;
  ret = savings(mdy(06,11,2017), mdy(07,11,2014), 500, 36, 'month', 'month',
               mdy(07,11,2014), 5, mdy(07,11,2015), 2);
  PUT 'The amount saved: ' ret nlmnlgbp12.0 ;
RUN;
```

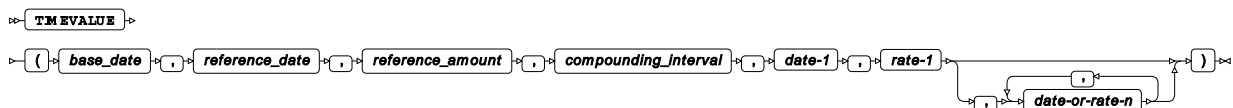
This produces the following output:

```
The amount saved: £18,152.63
```

In this example, the amount returned is less than that returned in the previous example as the interest rate decreased from 5% to 2% after a year.

TIMEVALUE

Returns the value of interest on savings plus the savings if interest is applied at a specified future date while the term of the principal remains the same.



For example, if you have access to a principal for five years, you can calculate the value of that principal with interest, if interest were to be applied six months after you had access to the principal.

Return type: Numeric

base_date

Type: Numeric

The date to which the money will be held.

reference_date

Type: Numeric

The date from which interest will be applied.

reference_amount

Type: Numeric

The principal to which the interest is applied.

compounding_interval

Type: Character

The interval at which compounding of interest occurs, such as monthly or quarterly.

An interval can be:

'DAY'

Compounding occurs at the end of each day.

'SEMIMONTH'

Compounding occurs at the middle and end of each month.

'MONTH'

Compounding occurs at the end of each months.

'QUARTER'

Compounding occurs at the end of each quarter year.

'SEMIYEAR'

Compounding occurs at the end of each half year.

'YEAR'

Compounding occurs at the end of each year.

date-1

Type: Numeric

The date at which the interest rate *rate-1* is valid.

rate-1**Type:** Numeric

The interest rate expected at *date-1* entered as a percentage, for example 5 for 5%.

date-or-rate-n

Optional argument

Type: Numeric

A pair of comma-separated values. The first is a date at which an interest rate *rate-n* becomes valid, and the second is the interest rate at that date. The date and interest rate must be specified as pairs. If the function finds unpaired values, an error occurs.

Dates are expected in numeric date format; if you want to specify dates in typical formats, such as 01 Jan 2017 or 09/11/2016, you will need to use a suitable function or format the data on input.

date-1 sets the date from which access to the principal begins. *date-1* must therefore be earlier than *base_date*.

Example – saved amount from specified time

In this example, the function calculates the amount saved, including interest, on a principal of £1,800 starting at the sixth months after savings could have started. Interest is compounded yearly. The result is written to the log.

```
DATA _NULL_;
  ret = timevalue(mdy(06,11,2019), mdy(12,11,2016), 18000, 'year',
    mdy(07,11,2016),5);
  PUT 'The amount saved: ' ret nlmnlgbp12.0 ;
RUN;
```

This produces the following output:

```
The amount saved: £20,340.87
```

Example – saved amount from specified time with changes to interest rate

In this example, the function calculates the amount saved, including interest, on a principal of £1,800 starting at the sixth months after savings could have started. Interest is compounded yearly. The interest rate decreases to 2% a year after the initial rate is set. The result is written to the log.

```
DATA _NULL_;
  ret = timevalue(mdy(06,11,2019), mdy(12,11,2016), 18000, 'year',
    mdy(07,11,2016),5,mdy(07,11,2018), 2);
  PUT 'The amount saved: ' ret nlmnlgbp12.0 ;
RUN;
```

This produces the following output:

```
The amount saved: £19,804.04
```

In this example, the amount returned is less than that returned in the previous example as the interest rate decreased from 5% to 2% after a year.

YIELDP

Returns the yield to maturity for a security for specified periodic cash flows.

YIELDP (*par-value* , *coupon-rate* , *numeric* , *K* , *k0* , *price*)

Return type: Numeric

par-value

Type: Numeric

The par value of the bond.

coupon-rate

Type: Numeric

The annual coupon rate.

numeric

Type: Numeric

The number of coupons per period. This might, for example, be one a year, or one a month, or twelve a year.

K

Type: Numeric

The number of remaining coupon payments.

k0

Type: Numeric

The time to the first coupon. This should be greater than 0 and less than $1/\text{numeric}$. For example, if coupons are monthly, and there are 28 days until the next coupon, you might enter this value as $28/30 * 1/12$, or as 0.078.

price

Type: Numeric

The current price of the bond.

If the values supplied to the function result in a yield to maturity greater than 1 or less than 0, an error message is returned.

Example

In this example, the yield to maturity is calculated for a bond with a par value of £100, a coupon rate of 5%, coupon payments twice a year with four further payments until expiry, and with a price of £95. There are two months until the next coupon payment. The result is written to the log and to the example dataset.

```
DATA _NULL_;
  yp= yieldp(100, (5/100),2,4,2/6*1/2,95);
  PUT 'The yield to maturity is: ' yp percent6.4;
RUN;
```

This produces the following output:

```
The yield to maturity is: 9.4%
```

Internet functions

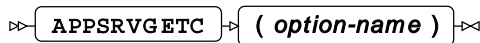
Send and receive information from internet-based resources. These functions can only be used in programs executed by an Application Server with a WPS Web application.

APPSRVGETC ↗	1579
Returns the value of an Application Server configuration setting.	
APPSRVGETN ↗	1580
Returns the value of an Application Server configuration setting.	
APPSRV_AUTHCLS ↗	1582
Returns a WHERE clause.	
APPSRV_AUTHDS ↗	1583
Modifies the dataset associated with authorisation functions.	
APPSRV_AUTHLIB ↗	1585
Tests access to individual items such as binaries, catalogs, datasets, and entities.	
APPSRV_HEADER ↗	1586
Modifies a list of HTTP headers according to set conditions.	
APPSRV_SESSION ↗	1587
The functions described in this section create or end a session for the current request, depending on the arguments provided.	
APPSRV_SET ↗	1588
Alters the value of the numeric setting of an Application Server for a request being executed.	
APPSRV_UNSAFE ↗	1590
Protects applications from code injection attacks through macro variables.	

APPSRVSET 	1591
The APPSRVSET function is an alias for the APPSRV_SET function. It alters the value of the numeric setting of an Application Server for a request being executed.	

APPSRVGETC

Returns the value of an Application Server configuration setting.



The name of the setting to be retrieved is provided as an argument to the function and must be present in the information below. If an invalid name is provided, an error is returned to the execution log. The names of settings are case insensitive.

Return type: Character

The returned type depends on the option specified.

option-name

Type: Character

The name of the setting to be retrieved.

The options are as follows:

charset

Specifies that the value of the Application Server `CHARSET` option is to be returned.
Returns an empty string if the `NOCHARSET` option was specified.

log file

Specifies that the path of the current log file being used by the Application Server is to be returned. The path will include any expanded substitution characters.

request init

Specifies that the name of the program an Application Server includes before every requested program, is to be returned. As specified in the `INIT` option of the `REQUEST` statement in the `APPSRV` procedure.

request term

Specifies that the name of the program an Application Server includes after every requested program, is to be returned. As specified in the `TERM` option of the `REQUEST` statement in the `APPSRV` procedure.

session invess

Specifies that the name of the program the Application Server executes when a user connects with an invalid session ID, is to be returned. As specified by the value of the `INVESS` option of the `SESSION` statement in the `APPSRV` procedure.

session term

Specifies that the name of the program that an Application Server executes when a session expires, is to be returned. As specified by the `TERM` option of the `SESSION` statement in the `APPSRV` procedure.

statistics data

Specifies that the name of the dataset used by the Application Server for recording request statistics, is to be returned.

version

Specifies that the version of the WPS software that the Application Server is using to execute the requested program, is to be returned.

Example

In this example, an HTTP response is returned that contains the version string of the WPS software running the Application Server.

```
DATA _NULL_;  
  FILE _WEBOUT;  
  version = APPSRVGETC('version');  
  PUT 'Content-type: text/plain';  
  PUT;  
  PUT version;  
RUN;
```

APPSRVGETN

Returns the value of an Application Server configuration setting.

```
>> APPSRVGETN ( option-name ) <<
```

This function is similar to `APPSRVGETC` but returns numeric values. If an invalid name is provided, an error is returned to the execution log. The names of settings are case insensitive.

Return type: Numeric

The value returned depends on the option specified.

option-name

Type: Character

The name of the setting to be retrieved.

The options are as follows:

automatic headers

Specifies that if the automatic header system is enabled for the program currently being executed, then 1 is returned, otherwise 0 is returned.

netbuffk

Specifies that the size of Application Server net buffer is to be returned. The buffer size is defined by the `NETBUFFK` option of the `PROC APPSRV` statement. The value returned is the size of the buffer in kilobytes.

port

Specifies that the port number the Application Server is listening on, is to be returned. As specified in the `PORT` option of the `PROC APPSRV` statement in the `APPSRV` procedure.

program error

Specifies the error code that will be returned by the `APPSRV_SET` function if a bad value is provided in the option-name.

request maxtimeout

Specifies that the maximum value of the `TIMEOUT` option of the `REQUEST` statement can take, to be returned. As specified by the `MAXTIMEOUT` option of the `REQUEST` statement in the `APPSRV` procedure.

request timeout

Specifies that the timeout value for program execution in seconds, before it is terminated by the Application Server, is to be returned. As specified by the `TIMEOUT` option of the `REQUEST` statement in the `APPSRV` procedure.

request read

Specifies that the timeout value in seconds for reading requests from a Broker, is to be returned. As specified by the `READ` option of the `REQUEST` statement in the `APPSRV` procedure.

server starttime

Specifies that the Coordinated Universal Time (UTC) timestamp of the exact time the Application Server started, is to be returned.

session timeout

Specifies that the timeout value of sessions in seconds is to be returned. As specified by the `TIMEOUT` option of the `SESSION` statement in the `APPSRV` procedure.

statistics writecount

Specifies that the number of statistics observations that are buffered before they are written to the Application Server's statistics dataset is to be returned. As defined by the `WRITECOUNT` option of the `STATISTICS` statement in the `APPSRV` procedure.

statistics writeevery

Specifies that the maximum delay in minutes before the statistics buffer is flushed to the Application Server statistics dataset, is to be returned. As defined by the `WRITEEVERY` option of the `STATISTICS` statement in the `APPSRV` procedure.

version

Specifies that the version number of the WPS software executing the program, is to be returned.

Example

In this example, an HTTP response is returned that contains the port number to which the Application Server is bound. The result is written to the log.

```
DATA _NULL_;  
  FILE _WEBOUT;  
  port = APPSRVGETN('port');  
  PUT 'Content-type: text/plain';  
  PUT;  
  PUT port;  
RUN;
```

APPSRV_AUTHCLS

Returns a WHERE clause.

➤ **APPSRV_AUTHCLS** ➤ (*type*) ➤

The WHERE clause can be used with metadata views, for example `dictionary`, to subset available resources corresponding to those authorised to be used by a program.

Return type: Character

type

Type: Character

The functions argument is the type of the item to be selected. This can be one of:

- MEMBER.
- CATALOGENTRY.
- LIBRARY.

Example

In this example, an HTML table is produced containing details of all datasets accessible according to the authorisation mechanism.

```
ODS HTML body=_webout path=&_tmpcat (url=&_replay) gpath=&_tmpcat (url=&_replay);

DATA _NULL_;
  FILE _WEBOUT;
  rc = APPSRV_AUTHCLS('MEMBER');
  CALL SYMPUT('where', rc);
RUN;

PROC SQL;
  select * from dictionary.members WHERE &where AND Memtype='DATA';
QUIT;

ODS HTML CLOSE;
```

APPSRV_AUTHDS

Modifies the dataset associated with authorisation functions.

➤ **APPSRV_AUTHDS** ➤ (*dataset*) ➤

Alters the dataset that contains rules for authorisation functions such as APPSRV_CLS and APPSRV_AUTHLIB.

Return type: Numeric

dataset

Type: Character

Rules are specified on a per-row basis. The following determine whether a rule operates on a whitelist or blacklist, and what the target(s) of a rule (are).

Access control functions operate on collections of rules that are held as rows in a dataset as follows:

Rule

Specify either `INCLUDE` or `EXCLUDE` to provide a whitelist or blacklist. To be granted access, an object must be included and not excluded.

Libname

Specify the name of the library.

Memname

Specify the name of the member of the library.

Memtype

Specify the type of the member in the library, it can be:

- DATA.
- CATALOG.
- VIEW.
- MDDB.

Objname

If `Memtype` is CATALOG then this refers to the name of a catalog entry.

Objtype

If `Memtype` is CATALOG then this refers to the type of a catalog entry.

All name and type variables can be assigned to the wildcard *, applying the rule to multiple objects. For example, the following rule would include all datasets in the SASHELP library:

Rule	Libname	Memname	Memtype	Objname	Objtype
INCLUDE	SASHELP	*	DATA	*	*

A default list of access control rules is provided in the SASHELP.AUTHLIB dataset, which can be viewed in the Workbench.

The APPSRV_AUTHDS DATA step function can be used to specify the name of a dataset containing a new set of rules. If the dataset is correctly formatted and loaded, the previous rules will be discarded.

Rules are discarded at the end of a requested program. If the authorisation mechanism is required for main programs in an application, the APPSRV_AUTHDS function could be used in a program specified by the INIT option of the REQUEST statement of the APPSRV procedure. This function creates Application Servers, ensuring that rules are automatically loaded for all requests.

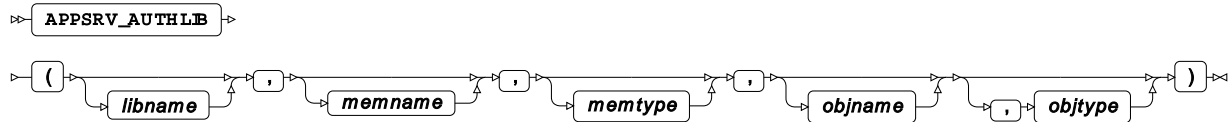
Example

This example illustrates how to load authorisation dataset rules from the dataset SECURITY.AUTHDS. If the rules cannot be loaded, then the program will exit:

```
DATA _NULL_;  
  rc = APPSRV_AUTHDS('security.authds');  
  IF rc ~= 1 THEN DO;  
    ENDWPS;  
  END;  
RUN;
```

APPSRV_AUTHLIB

Tests access to individual items such as binaries, catalogs, datasets, and entities.



Returns 1 or 0 to indicate if access is allowed or not, respectively.

Return type: Numeric

libname

Optional argument

Type: Character

memname

Optional argument

Type: Character

memtype

Optional argument

Type: Character

objname

Optional argument

Type: Character

objtype

Optional argument

Type: Character

Example

All options can be given the value * which acts as a wildcard and matches any value. The object name and object type options only apply if the item type option is equal to CATALOG as they specify a catalog entry.

The following DATA step sets the value of rc to 1 if access is granted to the PROTECTED library:

```
DATA _NULL_;  
  rc = APPSRV_AUTHLIB("PROTECTED", "*", "*", "*", "*");  
RUN;
```

The next example sets the value of `rc` to 1 if access is granted to any dataset in the `PROTECTED` library:

```
DATA _NULL_;
  rc = APPSRV_AUTHLIB('PROTECTED', "*", "DATA", "*", "*");
RUN;
```

In this example, the `DATA` step only executes if access is granted to the `DATALIB.PRIVATE` dataset:

```
%MACRO checkAccess;
  %IF %sysfunc(APPSRV_AUTHLIB(DATALIB, PRIVATE, DATA, *, *)) %THEN %DO;
    DATA _NULL_;
    SET DATALIB.PRIVATE;
    FILE _WEBOUT;
    PUT name;
  RUN;
%END;
%MEND;
%checkAccess
```

In this example a program will only execute the `CATALOGS` procedure if access is granted to the `SECRET.SOURCE` entry in the `SHARED` catalog of the `DATALIB` library.

```
%MACRO checkAccess;
  %IF %sysfunc(APPSRV_AUTHLIB(DATALIB, SHARED, CATALOG, SECRET, SOURCE)) %THEN %DO;
    PROC CATALOGS ...;
    ...
  RUN;
%END;
%MEND;
%checkAccess
```

APPSRV_HEADER

Modifies a list of `HTTP` headers according to `set` conditions.

```
➤ APPSRV_HEADER ➤ ( name , value ) ➤
```

Alters the list of `HTTP` headers that are prepended to the Application Server's response if it detects that the program itself did not generate any valid headers. Headers are specified as name-value pairs and override existing values if previously set by the program. Names cannot contain the colon (:), because this is the delimiter character in `HTTP`.

Return type: Character

name

Type: Character

value

Type: Character

Example

In this example, the program sets the *content-type* header for the response to `image/gif`, then writes an image in the body of the HTTP message. The step using the `GCHART` procedure is not fully specified.

```
DATA _NULL_;  
  rc = APPSRV_HEADER('Content-type', 'image/gif');  
RUN;  
PROC GCHART ...
```

The following example shows how to allow users to download a CSV file containing the contents of the `WORK.OUTPUT` dataset rather than displaying its contents in a browser:

```
DATA _NULL_;  
  FILE _WEBOUT TERMSTR=CRLF;  
  rc = APPSRV_HEADER('Content-type', 'text/csv');  
  rc = APPSRV_HEADER('Content-disposition', 'attachment; filename=output.csv');  
RUN;  
  PROC EXPORT DATA=WORK.OUTPUT OUTFILE=_webout DBMS=CSV REPLACE;  
RUN;
```

APPSRV_SESSION

The functions described in this section create or end a session for the current request, depending on the arguments provided.

CREATE Command

Creates a session for the current request.

➤ **APPSRV_SESSION** ➤ ("create" , session-timeout) ➤

Create

Create a new session for the current request using the Application Server's default timeout setting. If the request is associated with an existing session, or if the program has already created a session, then an error is returned.

An example function call:

```
DATA _NULL_;  
  id = APPSRV_SESSION('create');  
  PUT id;  
RUN;
```

Return type: Numeric

session-timeout

Type: Numeric

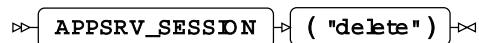
Create a new session with a timeout of n seconds. The timeout argument must be greater than 0 and less than the `session maxtimeout` setting, or an error is returned.

An example function call:

```
DATA _NULL_;  
  id = APPSRV_SESSION('create',45);  
  PUT id;  
RUN;
```

DELETE Command

Ends a session for the current request.



This marks the current session for deletion when the program finishes execution and then returns 0. If there is no session for the current request then the value of program error is returned.

An example function call:

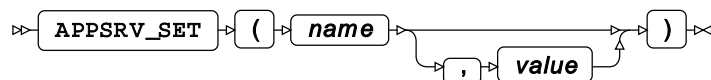
```
DATA _NULL_;  
  id = APPSRV_SESSION('delete');  
  PUT id;  
RUN;
```

Return type: Numeric

Either 0 or value of program error.

APPSRV_SET

Alters the value of the numeric setting of an Application Server for a request being executed.



This function can be used, for example, to:

- Disable automatic header generation
- Increase the maximum time a program can execute.
- Put an Application Server into a background mode.

Return type: Numeric

name

Type: Character

value

Optional argument

Type: Numeric

This function uses one or two arguments, both of which are case insensitive.

One argument

The following settings take one argument:

disconnect

Closes the network connection to the Broker, ending the response for a request but allowing an Application Server to continue processing the request itself.

background

Attempts to place the server into a background mode/state.

Two arguments

The following settings take two arguments:

session timeout

Sets the default session timeout if one is created during the processing of the request. This setting does not affect sessions that have already been created.

automatic headers

Enables the automatic header system if the second argument is greater than 0. Otherwise, it disables it.

requested timeout

Sets the total number of seconds that the current program will be allowed to execute for. The maximum allowable value that can be set can be obtained through `APPSRVGETN ('request maxtimeout')`.

background

```
APPSRV_SET('background',n);
```

does the same as:

```
APPSRV_SET('request timeout',n);  
APPSRV_SET('background');
```

program error

Sets the error code returned by `APPSRV_SET` if it is given invalid values.

If APPSRV_SET is successful, it returns 0. Otherwise, a program error is returned. By default, program error is set to 1.

Example

In this example, the program demonstrates how to disable the automatic header system for the remainder of program execution:

```
DATA _NULL_;
  rc = APPSRV_SET('automatic headers', 0);
RUN;
```

In the next example, the program disconnects the Application Server from the Broker, allowing the Broker to process the response while the server continues to execute the requested program:

```
DATA _NULL_;
  rc = APPSRV_SET('disconnect');
RUN;
```

In the last example, a DATA step places an Application Server into background mode, unbinding it from the TCP socket that it was using. If the server is not part of a pool service, or if the Load Manager does not allow backgrounding, then the rc variable is set to 1 and is used to display an error message instead of running the long task:

```
DATA _NULL_;
  rc = APPSRV_SET('background');
  IF rc ~= 0 THEN DO;
    PUT 'Error: Can't continue processing!';
    PUT 'Unable to place application server into background mode';
  END;
  ELSE;
    * Some long task;
    ...
  END;
RUN;
```

APPSRV_UNSAFE

Protects applications from code injection attacks through macro variables.

➤ **APPSRV_UNSAFE** ➤ (*paramname*) ➤

For protection, the Application Server automatically removes certain characters from the input parameters. These characters are specified in a default unsafe list or the UNSAFE option of the APPSRV procedure.

Return type: Character

paramname**Type:** Character

The original value of the input parameter (including any unsafe characters) can be retrieved using the APPSRV_UNSAFE DATA step function and used as a DATA step variable. The function takes a single argument that is the name of the parameter to retrieve and is case insensitive (unlike parameters in the HTTP protocol). If no parameter matches the provided name, an error is returned.

Note:

To reduce the risk of code injection, the returned value should not be assigned to a macro variable unless it is made safe through the macro quoting process.

Example

In this example, the program displays the original value of the `rawparam` parameter prior to the removal of any unsafe characters:

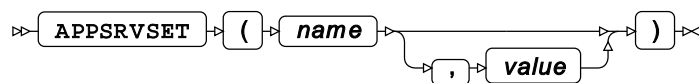
```
DATA _NULL_;  
  FILE _WEBOUT;  
  rawparam = APPSRV_UNSAFE('rawparam');  
  PUT 'Content-type: text/plain';  
  PUT;  
  PUT rawparam;  
RUN;
```

Note:

If the `rawparam` parameter contains control characters, then code is injected into the DATA step.

APPSRVSET

The APPSRVSET function is an alias for the APPSRV_SET function. It alters the value of the numeric setting of an Application Server for a request being executed.



This function can be used, for example, to:

- Disable automatic header generation
- Increase the maximum time a program can execute.
- Put an Application Server into a background mode.

Return type: Numeric

name

Type: Character

value

Optional argument

Type: Numeric

This function uses one or two arguments, both of which are case insensitive.

One argument

The following settings take one argument:

disconnect

Closes the network connection to the Broker, ending the response for a request but allowing an Application Server to continue processing the request itself.

background

Attempts to place the server into a background mode/state.

Two arguments

The following settings take two arguments:

session timeout

Sets the default session timeout if one is created during the processing of the request. This setting does not affect sessions that have already been created.

automatic headers

Enables the automatic header system if the second argument is greater than 0. Otherwise, it disables it.

requested timeout

Sets the total number of seconds that the current program is allowed to execute for. The maximum allowable value that can be set can be obtained through `APPSRVGETN ('request maxtimeout')`.

background

```
APPSRVSET('background',n);
```

does the same as:

```
APPSRVSET('request timeout',n);  
APPSRVSET('background');
```

program error

Sets the error code returned by `APPSRVSET` if it is given invalid values.

If `APPSRVSET` is successful, it returns 0. Otherwise, a program error is returned. By default, program error is set to 1.

Example

In this example, the program demonstrates how to disable the automatic header system for the remainder of program execution:

```
DATA _NULL_;
  rc = APPSRVSET('automatic headers', 0);
RUN;
```

In this example, the program disconnects the Application Server from the Broker, allowing the Broker to process the response while the server continues to execute the requested program:

```
DATA _NULL_;
  rc = APPSRVSET('disconnect');
RUN;
```

In this example, a DATA step places an Application Server into background mode, unbinding it from the TCP socket that it was using. If the server is not part of a pool service, or if the Load Manager does not allow backgrounding, then the rc variable is set to 1 and is used to display an error message instead of running the long task:

```
DATA _NULL_;
  rc = APPSRVSET('background');
  IF rc ~= 0 THEN DO;
    PUT 'Error: Can't continue processing!';
    PUT 'Unable to place application server into background mode';
  END;
  ELSE;
    * Some long task;
    ...
  END;
RUN;
```

List functions and CALL routines

Manipulate variables in lists. Lists contain values identified either by their position in the list, or by names.

CLEARLIST ↗	1596
Clear the items from a list, and optionally clear or delete sublists from it.	
COMPARELIST ↗	1597
Returns a value indicating whether the specified lists are identical.	
COPYLIST ↗	1601
Copies one list into another list.	
CURLIST ↗	1607
Set the specified list as the current list.	
DELITEM ↗	1612
Deletes the item at the specified position in a list.	

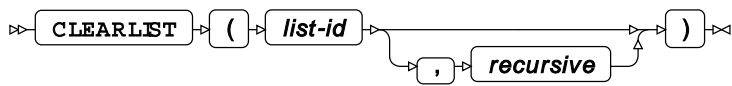
DELLIST ↗	1613
Deletes a list.	
DELNITEM ↗	1617
Deletes the specified named item and its associated value from a list.	
DESCRIBE ↗	1621
Enables a list to be populated with dataset, data view and catalog entry attributes.	
ENVLIST ↗	1625
Creates a list.	
FILLIST ↗	1627
Creates a list using the items in a previously saved list.	
GETITEMC ↗	1629
Returns the character or string at the specified position in the list.	
GETITEML ↗	1630
Returns the list identifier of the sublist at a specified position in a list.	
GETITEMN ↗	1631
Returns the number at the specified position in the list.	
GETLATTR ↗	1632
Returns the attributes that are set for a list or for a list item.	
GETLCNTA ↗	1634
Returns the number of lists that remain open.	
GETLCNTP ↗	1635
Returns the total number of lists that resulted from create and delete operations while <code>CALL LISTPROF</code> is active.	
GETNITEMC ↗	1636
Returns the character or string specified for a named list item.	
GETNITEML ↗	1641
Gets the sublist identifier held by the specified named list item.	
GETNITEMN ↗	1646
Returns the number specified for a named list item.	
HASATTR ↗	1650
Returns a value indicating whether a list or list item has a specified attribute value.	
INSERTC ↗	1654
Inserts a character or string at a specified position in the list.	
INSERTL ↗	1656
Insert a sublist into the specified list.	
INSERTN ↗	1659
Inserts a number at a specified position in a list.	
ITEMTYPE ↗	1662
Return the type of the item at a specified position in a list.	

LISTLEN ↗	1664
Returns the number of items in a list.	
LVARLEVEL ↗	1665
Creates a list using the values of a variable in a specified dataset.	
MAKELIST ↗	1667
Creates a list in which items are specified by ordinal position.	
MAKENLIST ↗	1668
Creates a list in which items are specified by variable name.	
NAMEDITEM ↗	1669
Returns the position of a named item in a list.	
NAMEITEM ↗	1672
Name or rename an item, or return the item name.	
POPC ↗	1675
Returns and removes a character or string item from a specified list.	
POPL ↗	1677
Returns and removes a sublist from a specified list.	
POPNI ↗	1679
Returns and removes a numeric item from a specified list.	
REVLIST ↗	1681
Reverses the order of the items in a list.	
ROTLIST ↗	1682
Rotates the order of the items in a numeric list by a specified number of characters.	
SAVELIST ↗	1683
Saves a list.	
SEARCHC ↗	1688
Returns the ordinal position of a string in a list that matches the specified character or string.	
SEARCHL ↗	1692
Returns the position of a specified sublist in a list.	
SEARCHN ↗	1694
Returns the ordinal position of a specified number.	
SETITEMC ↗	1696
Sets the specified list item to the specified character string.	
SETITEML ↗	1699
Sets the specified list item to the specified sublist.	
SETITEMN ↗	1703
Sets the specified list item to the specified number.	
SETLATR ↗	1706
Set the attributes for a list or for specified items in a list.	
SETNITEMC ↗	1712
Sets the specified named list item to the specified character string.	

SETNITEML ↗	1719
Sets the specified list item to the specified sublist.	
SETNITEMN ↗	1725
Sets the specified named list item to the specified number.	
SORTLIST ↗	1732
Sorts the specified list.	
CALL LISTPROF ↗	1735
Enables the number of lists created in a DATA step to be calculated.	
CALL PUTLIST ↗	1737
Write the specified list to the log.	

CLEARLIST

Clear the items from a list, and optionally clear or delete sublists from it.



If the list contains sublists, these are not removed unless requested.

Return type: Numeric

list-id

Type: List

The identifier for the list.

recursive

Optional argument

Specifies how to handle sublists.

"N"

Sublists are not cleared or deleted. This is the default.

"Y"

Sublists are cleared. The contents of the sublist are deleted, but the identifier is not deleted.

"D"

Sublists are cleared and the identifier for the sublist is deleted. Other functions will no longer be able to access the sublist.

Example

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMC](#) (page 1696) functions. The list is then cleared. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST('g', 'v1', 'v2', 'v4');

  lr = SETNITEMC(lid1, 'plane', 'v1');
  lr = SETNITEMC(lid1, 'ship', 'v2');
  lr = SETNITEMC(lid1, 'horse', 'v4');

  cl = CLEARLIST(lid1);

  DO y = 1 TO listlen(lid1);
    gr = GETITEMC(lid1, y);
    PUT gr;
  END;

  ll = LISTLEN(lid1);
  PUT ll=;

RUN;
```

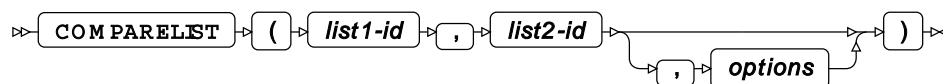
This produces the following output:

```
ll=0
```

The `GETITEMC` in the `DO` loop produces no output, because the length of the list is 0 (zero) and the loop therefore never starts.

COMPARELIST

Returns a value indicating whether the specified lists are identical.



Two specified lists are compared. Without options, the lists must be exactly the same. For lists with unnamed items, this means that the values must be the same, and occupy the same position in the list: `A B C` is different to `B A C`. For named lists, named items must match, and have the same values: `li1=1 li2=3 li3=4` is the same as `li1=1 li3=4 li2=3`; however, it is different to `li1=4 li2=3 li3=1`. By default, case is ignored; for example, `Horse`, `HORSE` and `horse` are all identical.

Options enable you to modify the comparison, so that, for example, case is considered (`Horse`, `HORSE` and `horse` would be the same), or item names or item values are ignored in the comparison.

Return type: Numeric

0 (zero) if the lists match; 1 otherwise.

list1-id

Type: List

The identifier of the first list to be compared.

list2-id

Type: List

The identifier of the second list to be compared.

options

Optional argument

One or more options, enclosed in quotation marks. If more than one option is specified, separate options with spaces; for example, 'NAME MIXEDCASE NODUMP'.

The following options are available:

"NAME"

Item names are considered in the comparison.

"NONAME"

Item names are ignored in the comparison.

"NOHONORCASE"

If the `HONORCASE` attribute has been set for a list, the attribute is ignored in the comparison.

"MIXEDCASE"

Case is taken into account in the comparison. For example, if you do not specify this option, `CAR` and `car` are identical; if you do specify this option, they are different.

"ITEM"

Item values are considered in the comparison.

"NOITEM"

Item values are ignored in the comparison.

"NODUMP"

No information about the comparison is written to the log. This is the default.

"LONGDUMP"

All information about the comparison is written to the log.

"SHORTDUMP"

Information about the comparison of up to five items is written to the log.

The results of comparing lists can be unexpected if the lists to be compared have been created in different ways (for example, using `MAKELIST` and `MAKENLIST`), or list attributes such as `'HONORCASE'` have been set.

Basic example

In this example, two lists are created. Named items are entered into each list. These lists are then compared. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1','var2','var3');

  lr1 = SETNITEMC(lid1, 'car',      'var1');
  lr1 = SETNITEMC(lid1, 'bicycle',  'var2');
  lr1 = SETNITEMC(lid1, 'train',    'var3');

  lid2 = MAKENLIST('g', 'var1','var2','var3');

  lr2 = SETNITEMC(lid2, 'car',      'VAR1');
  lr2 = SETNITEMC(lid2, 'bicycle',  'VAR2');
  lr2 = SETNITEMC(lid2, 'train',    'VAR3');

  IF COMPARELIST(lid1,lid2) EQ 0 THEN PUT 'Lists are the same';
      ELSE PUT 'Lists are not the same';

RUN;
```

This produces the following output:

```
Lists are the same
```

The lists have the same item values, and the same item names, and because case is ignored, the lists are therefore regarded as identical.

Example – accounting for case in values, and writing results to log

In this example, two lists are created. Named items are entered into each list. These lists are then compared; the 'MIXEDCASE' option is set, which specifies that the case of item values should be considered in the compare. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1','var2','var3');

  lr1 = SETNITEMC(lid1, 'car',      'var1');
  lr1 = SETNITEMC(lid1, 'bicycle',  'var2');
  lr1 = SETNITEMC(lid1, 'train',    'var3');

  lid2 = MAKENLIST('g', 'VAR1','VAR2','VAR3');

  lr2 = SETNITEMC(lid2, 'CAR',      'var1');
  lr2 = SETNITEMC(lid2, 'bicycle',  'var2');
  lr2 = SETNITEMC(lid2, 'train',    'var3');

  IF COMPARELIST(lid1, lid2, 'MIXEDCASE SHORTDUMP') EQ 0 THEN PUT 'Lists are the
same';
      ELSE PUT 'Lists are not the same';

RUN;
```

This produces the following output:

```
NOTE: List item 1: in list 1 NAME='VAR1', TYPE='C', VALUE='car', in list 2
      NAME='VAR1',
          TYPE='C', VALUE='CAR'
Lists are not the same
```

The values `CAR` and `car` differ only in case. If 'MIXEDCASE' had not been set, the lists would have been regarded as identical. The log shows where the lists differ.

Example – ignoring variable names

In this example, two lists are created. Named items are entered into each list. These lists are then compared; the 'NONAME' option is set, which specifies that item names are not compared. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST();

  lr1 = SETNITEMC(lid1, 'car',      'var1');
  lr1 = SETNITEMC(lid1, 'bicycle', 'var2');
  lr1 = SETNITEMC(lid1, 'train',   'var3');

  call putlist(lid1);

  lid2 = MAKENLIST();

  lr2 = SETNITEMC(lid2, 'CAR',      'var2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'var5');
  lr2 = SETNITEMC(lid2, 'train',   'var3');

  call putlist(lid2);

  IF COMPARELIST(lid1, lid2, 'NONAME') EQ 0 THEN PUT 'Lists are the same';
  ELSE PUT 'Lists are not the same';

RUN;
```

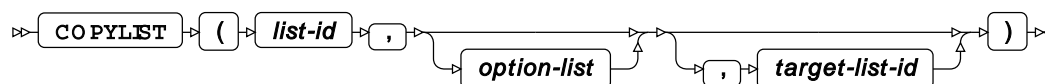
This produces the following output:

```
Lists are the same
```

The item names have been ignored, and the case is ignored by default. The lists are, therefore, the same.

COPYLIST

Copies one list into another list.



Items are copied from the first specified list (the source list) and appended to the second specified list (the target list). Various options enable you to specify how the items in sublists will be copied.

Return type: List

list-id

Type: List

The list identifier of the list (the source list) you want to copy to another list. All items in this list are appended to the target list, unless you set options *option-list* that specify otherwise.

option-list

Optional argument

Specifies how sublists are copied, and whether lists are merged.

"N"

The list identifier of any source list is copied into the target list (that is, a shallow copy is performed). This is the default.

"NO"

The list identifier of any source list is copied into the target list (that is, a shallow copy is performed). This is the default.

"NONRECURSIVELY"

The list identifier of any source list is copied into the target list (that is, a shallow copy is performed). This is the default.

"M"

If an item name is the same in both lists, then the value of the item in the source list overwrites the value of the item with the same name in the target list.

For example, suppose the source list has items named v_1 , v_2 , and v_3 , while the target list has the variables v_1 , v_2 and v_4 . The items v_1 and v_2 will be overwritten in the target list, v_3 will be not be overwritten and retain its current value, while v_4 will be appended to the list.

If the target list is not a list composed of named items, the source list is appended at the end of the target list. If the source list is not a named list, the items are not appended to the target list and an error message is returned for each unnamed item. If both lists contain items specified by position (that is, neither list contains named list items), then items in the source list are not appended to the target list and an error message is returned for each unnamed item.

"MERGE"

If an item name is the same in both lists, then the value of the item in the source list overwrites the value of the item with the same name in the target list.

For example, suppose the source list has items named v_1 , v_2 , and v_3 , while the target list has the variables v_1 , v_2 and v_4 . The items v_1 and v_2 will be overwritten in the source list, v_3 will be not be overwritten and retain its current value, while v_4 will be appended to the list.

If the target list is not a list composed of named items, the source list is appended at the end of the target list. If the source list is not a named list, the items are not appended to the target list and an error message is returned for each unnamed item. If both lists contain items specified by position (that is, neither list contains named list items), then items in the source list are not appended to the target list and an error message is returned for each unnamed item.

"Y"

A new list and list identifier is created for any source list copied into the target list (that is, a deep copy is performed).

"YES"

A new list and list identifier is created for any source list copied into the target list (that is, a deep copy is performed).

"RECURSIVELY"

A new list and list identifier is created for any source list copied into the target list (that is, a deep copy is performed).

target-list-id

Optional argument

Type: List

The identifier of the list (the target list) to which the items in *list-id* (the source list) are copied. If *target-list-id* already contains items, then the items in *list-id* are appended to it, unless otherwise directed by options set in *option-list*.

If this option is not specified, *list-id* is not copied to any list.

An indexed list can be copied to a named list, and vice versa, unless *option-list* is set to 'MERGE'.

Basic example

In this example, two lists are created using the `MAKENLIST` [\(page 1667\)](#) and `SETNITEMC` [\(page 1696\)](#) functions. `COPYLIST` is then used to append one list to the end of the other list. The items are then extracted from the target list using `GETITEMC` [\(page 1629\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'v1', 'v2', 'v3');

  lr1 = SETNITEMC(lid1, 'car', 'v1');
  lr1 = SETNITEMC(lid1, 'bicycle', 'v2');
  lr1 = SETNITEMC(lid1, 'train', 'v3');

  lid2 = MAKENLIST('g', 'v1', 'v2', 'v3');

  lr2 = SETNITEMC(lid2, 'plane', 'v1');
  lr2 = SETNITEMC(lid2, 'ship', 'v2');
  lr2 = SETNITEMC(lid2, 'horse', 'v3');

  clr = COPYLIST(lid2,,lid1);

  DO y = 1 TO LISTLEN(lid1);
    gi = GETITEMC(lid1, y);
    PUT gi;
  END;

RUN;
```

This produces the following output:

```
car
bicycle
train
plane
ship
horse
```

Example – copy with merge

In this example, two lists are created using the [MAKELIST](#) (page 1667) and [SETNITEMC](#) (page 1696) functions. [COPYLIST](#) is then used to append one list to the end of the other list. The items are then extracted from the target list using [GETITEMC](#) (page 1629). The [MERGE](#) option is specified so that the two lists are merged. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST('g', 'v1', 'v2', 'v3');

  lr1 = SETNITEMC(lid1, 'car', 'v1');
  lr1 = SETNITEMC(lid1, 'bicycle', 'v2');
  lr1 = SETNITEMC(lid1, 'train', 'v3');

  lid2 = MAKELIST('g', 'v1', 'v2', 'v4');

  lr2 = SETNITEMC(lid2, 'plane', 'v1');
  lr2 = SETNITEMC(lid2, 'ship', 'v2');
  lr2 = SETNITEMC(lid2, 'horse', 'v4');

  clr = COPYLIST(lid2, 'MERGE', lid1);

  DO y = 1 TO LISTLEN(lid1);
    gi = GETITEMC(lid1, y);
    PUT gi;
  END;

RUN;
```

This produces the following output:

```
plane
ship
train
horse
```

In this example, both lists contain items named `v1` and `v2`. The values for `v1` and `v2` in the source list (`lid2`) overwrite the values for `v1` and `v1` in the target list (`lid1`). Therefore, `plane` and `ship` overwrite `car` and `bicycle`.

The source list does not contain a list item named `v3`, so that item in the target list is not overwritten. The target list does not contain an item named `v4`, so that item from the source list is appended to the target list.

Example – copy with **RECURSIVELY** specified

In this example, three lists are created. One of these lists is inserted into another list. [COPYLIST](#) is then used to copy the list containing a sublist into another. The `'RECURSIVELY'` option is set, which will create a new list with its own identifier. The items are then extracted from the target list using [GETITEMC](#) (page 1629) and [GETITEML](#) (page 1629). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(3);
```

```
lr1 = SETITEMC(lid1, 'car', 1);
lr1 = SETITEMC(lid1, 'bicycle', 2);
lr1 = SETITEMC(lid1, 'train', 3);

lid2 = MAKELIST(3);

lr2 = SETITEMC(lid2, 'seal', 1);
lr2 = SETITEMC(lid2, 'whale', 2);
lr2 = SETITEMC(lid2, 'bat', 3);

lid3 = MAKELIST(3);

lr3 = SETITEMC(lid3, 'carrot', 1);
lr3 = SETITEMC(lid3, 'cabbage', 2);
lr3 = SETITEMC(lid3, 'lettuce', 3);

il = INSERTL(lid2, lid3, 3);

cl = COPYLIST(lid2, 'RECURSIVELY', lid1);

DO y = 1 TO LISTLEN(lid1);
  IF ITEMTYPE(lid1, y) EQ 'C' THEN
    DO;
      gi = GETITEMC(lid1, y);
      PUT gi;
    END;

    IF ITEMTYPE(lid1, y) EQ 'L' THEN
      DO;
        gi = GETITEML(lid1, y);
        PUT gi;
      END;
    END;
  END;

nl = GETLCNTA();

PUT 'Number of lists created: ' rc;

RUN;
```

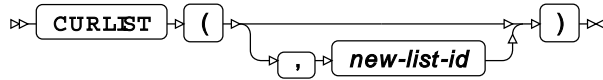
This produces the following output:

```
car
bicycle
train
seal
whale
4
bat
Number of lists created: 4
```

After the recursive copy, the identifier of the inserted list is 4, rather than the 3 that would have been assigned to it when created. GETLCNTA has been used to count the number of lists created during the session. This function has also returned 4, showing that when the list *lid2* was copied to list *lid1*, another copy of the sublist identified by *lid3* was created. The total number of lists created is now, therefore, four.

CURLIST

Set the specified list as the current list.



The list identifier is held in memory and can be made available to other functions. The list identifier can then be used as a default identifier in some functions.

Note:

The only function that currently exploits `CURLIST` is `LVARLEVEL`.

Return type: List

0 (zero), or a list identifier.

new-list-id

Optional argument

Type: List

The identifier of the list you want to make the current list.

The value returned depends on whether you specify *new-list-id*:

- If you do specify *new-list-id*, the identifier for the previous current list is returned. If the list you are making current is the first (that is, its identifier is 1), then 0 (zero) is returned.
- If you do not specify *new-list-id*, the identifier of the current list is returned.

You can use the value returned for the previous current list to swap between lists in other functions. See the example in the section *Example – specifying and not specifying new-list-id* below for more details.

Basic example

In this example, a new list is created that has the same size as the number of observations in the specified dataset. `CURLIST` is used to make the identifier of the list the current identifier. `LVARLEVEL` is then used to fill a list with the observations associated with a specified variable. No list identifier is specified to `LVARLEVEL`, as it can access the current identifier. The result is written to the log.

```
LIBNAME lbooks 'c:\temp';
DATA _NULL_;

  var = 0;
  did = OPEN('lbooks.books');

  lid = ENVLIST('g');

  cl = CURLIST(lid);

  lvl = LVARLEVEL(did, 'Author', var);

  DO i = 1 TO var;
    gi = GETITEMC(lid, i);
    PUT gi;
  END;

RUN;
```

This produces the following output:

```
Bruce, Steve
Ritzer, George
Cohen, I B
Carey, Nessa
Gribbin, John
Davies, Paul
Humphrey, Nicholas
Honderich, Ted
Grayling, A C
Buchanan, R A
Ellul, Jacques
Hindle, Paul
Marwick, Arthur
Hibbert, Christopher
Rostow, W W
Aldgate, Anthony, et al
Williams, Raymond
Hodgkiss, Phillip
```

Example – persistence of the current list

In this example, a new list is created that has the same size as the number of observations in the specified dataset. CURLIST is used to make the identifier of the list the current identifier. LVARLEVEL is then used twice. No list identifier is specified to LVARLEVEL, as it can access the current identifier each time it is used. The result is written to the log.

```
LIBNAME lbooks 'c:\temp';
DATA _NULL_;

var = 0;
did = OPEN('lbooks.books');

lid = ENVLIST('g');

cl = CURLIST(lid);

lvl = LVARLEVEL(did, 'Author', var);

PUT 'Authors: ';

DO i = 1 TO var;
    gia = GETITEMC(lid, i);
    PUT gia;
END;

PUT ' ';

lvl = LVARLEVEL(did, 'Title', var);

PUT 'Titles: ';

DO i = 1 TO var;
    git = GETITEMC(lid, i);
    PUT git $78.;
END;

RUN;
```

This produces the following output:

```
Authors:
Bruce, Steve
Ritzer, George
Cohen, I B
Carey, Nessa
Gribbin, John
Davies, Paul
Humphrey, Nicholas
Honderich, Ted
Grayling, A C
Buchanan, R A
Ellul, Jacques
Hindle, Paul
Marwick, Arthur
Hibbert, Christopher
Rostow, W W
Aldgate, Anthony, et al
Williams, Raymond
```

Hodgkiss, Phillip

Titles:

Sociology: A Very Short Introduction

Introduction to Sociology

Birth of the New Physics, The

The Epigenetics Revolution: How Modern Biology is Rewriting Our Understanding

In Search of the Edge of Time

About Time

History of the Mind, A

Oxford Companion to Philosophy

Truth, Meaning and Realism

Power of the Machine, The

Technological Society

Medieval Roads and Tracks

Sixties, The

British Society Since 1945

Cities and Civilizations

Stages of Economic Growth, The

Windows on the Sixties

Television: Technology and Cultural Form

Making of the Modern Mind, The

Example – specifying and not specifying *new-list-id*

In this example, two new lists are created. Two lists are set up with different identifiers, one to contain a list of authors, the other to contain a list of book titles. The result is written to the log.

```
LIBNAME lbooks 'c:\temp';
DATA _NULL_;

var = 0;
did = OPEN('lbooks.books');
no = ATTRN(did, 'nlobs');

lid1 = MAKELIST(no);

cl = CURLIST(lid1);

lvl = LVARLEVEL(did, 'Author', var);

gc1 = GETITEMC(lid1, 2);
PUT gc1;

lid2 = MAKELIST(no);

prev = CURLIST(lid2);
lvl = LVARLEVEL(did, 'Title', var);

gc2 = GETITEMC(lid2, 2);
PUT gc2;

gc3 = GETITEMC(prev, 3);
PUT gc3;

now = CURLIST();

gc4 = GETITEMC(now, 3);
PUT gc4;

RUN;
```

This produces the following output:

```
Ritzer, George
Introduction to Sociology
Cohen, I B
Birth of the New Physics, The
```

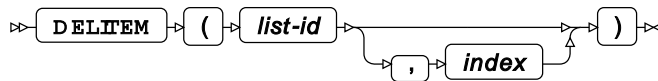
In this example, two lists are set up with different identifiers, one to contain a list of authors, the other to contain a list of book titles. `CURLIST` is used three times:

- The first time, the current list is set as the list with the first list identifier.
- The second time, the current list is set as the list with second list identifier. The function returns the value of the previous (in this case, the first) list identifier to the variable *prev*. This variable is used in a subsequent function to get the data from the first list.
- The third time, the function is used without the *new-list-id* argument, and so returns the value of the current list identifier to the variable *now*. This variable is then used in the subsequent function to get data from the second list.

The output is also returned in reverse order, as the order of the items in the list created by `LVARLEVEL` is the reverse of that in the dataset or input file.

DELITEM

Deletes the item at the specified position in a list.



Return type: List

The list identifier.

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of the item.

The default is 1.

The length of the list decreases by one after an item has been deleted.

Example

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. The result is written to the log. The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). This displays the original list. The second list item is then deleted, and the items are the extracted from the list again.

```
DATA _NULL_;

  lid1 = MAKELIST(3);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'bicycle', 2);
  lr1 = SETITEMC(lid1, 'train', 3);

  PUT 'Original list: ';

  DO y = 1 TO listlen(lid1);
    gi = GETITEMC(lid1, y);
    PUT gi;
  END;

  dr = DELITEM(lid1, 2);

  PUT ' ';
  PUT 'List after item deleted: ';

  DO y = 1 TO listlen(lid1);
    gi = GETITEMC(lid1, y);
    PUT gi;
  END;

RUN;
```

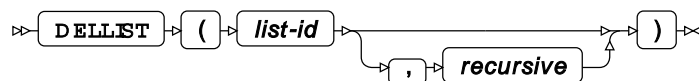
This produces the following output:

```
Original list:
car
bicycle
train

List after item deleted:
car
train
```

DELLIST

Deletes a list.



Although the specified list and its contents is deleted, the list identifier continues to exist. If you attempt to find the length of a deleted list using `LISTLEN`, `-1` is returned.

Return type: Numeric

The value assigned to the specified item name. If the named item does not contain a character, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

recursive

Optional argument

Specify whether to delete sublists.

"N"

Sublists are not deleted. This is the default.

"Y"

Sublists are deleted.

Basic example

In this example, a list is created using the `MAKENLIST` [\(page 1668\)](#) and `SETNITEMC` [\(page 1696\)](#) functions. The list is then cleared. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'v1', 'v2', 'v4');

  lr1 = SETNITEMC(lid1, 'plane', 'v1');
  lr1 = SETNITEMC(lid1, 'ship', 'v2');
  lr1 = SETNITEMC(lid1, 'horse', 'v4');

  dr = dellist(lid1);

  PUT lid1=;

RUN;
```

This produces the following output:

```
lid2=1
```

The contents of the list are deleted, but the identifier remains assigned after deletion.

Example – deleting list containing sublists

In this example, a list is created using the `MAKENLIST` [↗](#) (page 1668) and `SETNITEMC` [↗](#) (page 1696) functions. A sublist is then similarly created. The list is then cleared without specifying the *recursive* argument. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'v1', 'v2', 'v3');

  lr1 = SETNITEMC(lid1, 'car', 'v1');
  lr1 = SETNITEMC(lid1, 'bicycle', 'v2');
  lr1 = SETNITEMC(lid1, 'train', 'v3');

  lid2 = MAKENLIST('g', 'v1', 'v2', 'v4');

  lr2 = SETNITEMC(lid2, 'plane', 'v1');
  lr2 = SETNITEMC(lid2, 'ship', 'v2');
  lr2 = SETNITEMC(lid2, 'horse', 'v4');

  ir = insertl(lid1,lid2,3);

  CALL PUTLIST(lid1,,0);

  dr = dellist(lid1);

  PUT 'After deletion: ';

  CALL PUTLIST(lid2,,0);

RUN;
```

This produces the following output:

```
(V1='car'
 V2='bicycle'
 (V1='plane'
  V2='ship'
  V4='horse'
 ) [2]
 V3='train'
 ) [1]
After deletion:
NOTE: Argument 1 to function PUTLIST at line 49 column 8 is invalid
(V1='plane'
 V2='ship'
 V4='horse'
 ) [2]
_N_=1 _ERROR_=1 lid1=3 rc=0 lid2=4
```

The list identified by *lid1* has been deleted, and the attempt to write it to the log using `CALL PUTLIST` results in an error. The list identified by *lid2* is written, however, as *recursive* was not set to `Y`.

In the following DATA step, *recursive* is set to Y.

```
DATA _NULL_;

    lid1 = MAKENLIST('g', 'v1', 'v2', 'v3');

    rc = SETNITEMC(lid1, 'car', 'v1');
    rc = SETNITEMC(lid1, 'bicycle', 'v2');
    rc = SETNITEMC(lid1, 'train', 'v3');

    lid2 = MAKENLIST('g', 'v1', 'v2', 'v4');

    rc = SETNITEMC(lid2, 'plane', 'v1');
    rc = SETNITEMC(lid2, 'ship', 'v2');
    rc = SETNITEMC(lid2, 'horse', 'v4');

    rc = insertl(lid1, lid2, 3);

    CALL PUTLIST(lid1, , 0);

    rc = dellist(lid1, 'Y');

    PUT 'After deletion: ';

    CALL PUTLIST(lid1, , 0);
    CALL PUTLIST(lid2, , 0);

RUN;
```

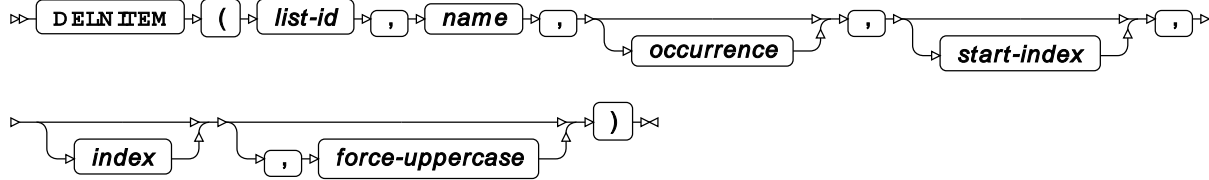
The following is output to the log:

```
(V1='car'
V2='bicycle'
(V1='plane'
V2='ship'
V4='horse'
)[6]
V3='train'
)[5]
After deletion:
NOTE: Argument 1 to function PUTLIST at line 76 column 8 is invalid
NOTE: Argument 1 to function PUTLIST at line 77 column 8 is invalid
_N_=1 _ERROR_=1 lid1=5 rc=0 lid2=6
```

Messages are returned for each attempt to write the list to the log, as both lists have been deleted.

DELNITEM

Deletes the specified named item and its associated value from a list.



If there is more than one occurrence of the same name, the value of the first is deleted, unless you specify which occurrence to find.

Return type: List

The list identifier.

list-id

Type: List

The identifier for the list.

name

Type: Character

The name of the item to delete.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

The ordinal position in the list at which to start searching for the named item from which you want the value. By default, the search starts at the first item. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

index

Optional argument

Type: Var

The name of a variable in which the position of the item specified is returned. If the value of this argument is anything except a variable name, an error is returned.

For example, if the item deleted is at the tenth position, 10 is returned.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, a new list is created that contains four named items. The list item named `var2` is then deleted. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMN(lid1, 1, 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, '300', 'var4');

  dr = DELNITEM(lid1, 'var2');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1=1
 VAR3='horse'
 VAR4='300'
 ) [1]
```

The output shows that `var2` and its value have been deleted from the list.

In this example, the case in which you specify the item names is ignored. You could, for example, have specified:

```
rc = SETNITEMN(lid1, 1, 'Var1');
lr1 = SETNITEMC(lid1, 'plane', 'var2');
lr1 = SETNITEMC(lid1, 'horse', 'var3');
lr1 = SETNITEMC(lid1, '300', 'VAR4');
```

The output would still be the same.

Example – non-unique item names

In this example, a new list is created that can hold six named items, where three of the items have non-unique item names. The second occurrence of the item named `var2` is then deleted. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMN(lid1, 1, 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, '300', 'var4');
  lr1 = SETNITEMC(lid1, 'taxi', 'var2', 2);
  lr1 = SETNITEMC(lid1, 'plane', 'var2', 3);

  dr = DELNITEM(lid1, 'var2', 2);

  CALL PUTLIST(lid1, , 0);

RUN;
```

This produces the following output:

```
(VAR1=1
 VAR2='plane'
 VAR3='horse'
 VAR4='300'
 var2='plane'
) [1]
```

If the occurrence is not specified, as in the following statement, the first occurrence of the item name is deleted:

```
s = DELNITEM(lid1, 'var2');
```

This produces the following output:

```
(VAR1=1
 VAR3='horse'
 VAR4='300'
 var2='taxi'
 var2='plane'
) [1]
```

Example – starting from an index position

In this example, a new list is created that can hold six named items, where three of the items have non-unique item names. The second occurrence of the list item named `var2`, starting from the fifth position in the list, is then deleted. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1', 'var2', 'var3', 'var4','var2', 'var2');

  lr1 = SETNITEMN(lid1, 1,      'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, '300',   'var4');
  lr1 = SETNITEMC(lid1, 'taxi',  'var2',2);
  lr1 = SETNITEMC(lid1, 'plane', 'var2',3);

  s = DELNITEM(lid1, 'var2',2,5);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1=1
VAR2='plane'
VAR3='horse'
VAR4='300'
var2='taxi'
)[1]
```

The third occurrence of the item named `var2` has been deleted because the function searched for the second occurrence of the item name starting from the fifth index position.

Example – specifying case

In this example, a new list is created that holds four named items. The names are specified in upper case. The value for each item is set using an equivalent lower case name in `SETNITEMC`. The value is then obtained using `GETNITEMC`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1', 'var2', 'var3', 'VAR4');

  ix = 0;
  sla = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMN(lid1, 1, 'var1',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'plane', 'var2',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'horse', 'var3',,,ix,'y');
  lr1 = SETNITEMC(lid1, '300', 'var4',,,ix,'y');

  dr1 = DELNITEM(lid1, 'var2',,,ix, 'y');
  dr2 = DELNITEM(lid1, 'var3');

  CALL PUTLIST(lid1,,0);

RUN;
```

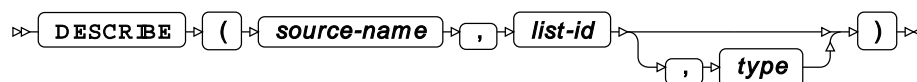
This produces the following output:

```
(NOTE: Argument to function DELNITEM at line 87 column 7 is invalid
(VAR1=1
VAR3='horse'
VAR4='300'
)[5]
_N_=1 _ERROR_=1 lid1=5 ix=2 mm=0 rc=5 s=.
```

The first use of `DELNITEMN` deletes the specified item `var2` as the case matches. The second use of `DELNITEMN` does not delete the specified value, as the case of the name is not forced to upper case. A message is written to the log, as no item name matches `var3`.

DESCRIBE

Enables a list to be populated with dataset, data view and catalog entry attributes.



This function requires a list to be created using one of the list creation functions, [MAKELIST](#) (page 1667) or [MAKENLIST](#) (page 1668). For datasets and dataset views, the list can then be populated with the dataset attributes returned by [ATTRC](#) (page 725) and [ATTRN](#) (page 727). For catalog entries the list can then be populated with attributes returned for the keywords described below.

Return type: Numeric

0 (zero) if successful.

source-name

Type: Character

The name of the dataset, data view or catalog entry for which you want information about attributes. The name must be specified using libname format, for example `lbooks.books`.

list-id

Type: List

The identifier of a list into which the attributes will be copied. The list must have item names that match the attributes to be returned. For example, if you want to return information about whether the dataset is indexed, the list must contain an item named `MODE`. See the section below for more information, and the examples.

type

Optional argument

Type: Character

The type of dataset. This can be:

'DATA'

A dataset.

'CATALOG'

A catalog entry.

'VIEW'

A data view.

If you do not specify a value for this argument, the type of dataset is determined by the number of filename elements that constitute *source-name*. For example, if you specify:

- `books` or `booklib.books`, the source is assumed to be a dataset or data view (in the first case, `work.books` is assumed).
- `booklib.books.history.catams`, the source is assumed to be a catalog entry. If you omit the last element of the name for a catalog entry (for example, `booklib.books.history`), the default is `program`.

Lists must be named lists. For datasets and data views, a list item name must be an attribute corresponding to one of the keywords listed under the attribute option of the `ATTRC` [↗](#) (page 725) and `ATTRN` [↗](#) (page 727) functions. For example, `LIB` returns as a string the library in which the dataset resides; `ANOBS` returns the value 1 or 0, depending on whether the number of observations is available.

`ATTRC` returns characters, and `ATTRN` returns a number, so the list item must be set with the appropriate function (`SETNITEMC` [↗](#) (page 1712) or `SETNITEMN` [↗](#) (page 1725)). See the examples below.

When you create a list item it can have any value, as the initial values will be replaced by the value of the corresponding attribute.

For catalog entries, a list item name can be one of the following:

'DESC'

The description provided for the catalog entry.

'EDESC'

The extended description provided for the catalog entry. Extended descriptions are not currently supported, so no value is returned.

'CRDATE'

The date at which the dataset entry was created.

'DATE'

The date at which the dataset entry was last modified.

'CRDATE' and 'DATE' can return the date as a string or as a number. If a string, the format is *dd/mmm/yy*, where *dd* is the numeric day in the month (for example, 11), *mmm* is an abbreviation for the English language name for the month (for example, SEP), and *yy* is the last two digits of the year (for example, 17). If a number, it is the number of days since epoch.

Basic example

In this example, attributes are returned from a specified dataset to a list. The resulting list is then written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
LIBNAME BOOK_DIR 'c:\temp\books';
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'LIB');
  lr1 = SETNITEMC(lid1, '', 'MODE');
  lr1 = SETNITEMN(lid1, ., 'LRECL');

  dsc = DESCRIBE('book_dir.books', lid1);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(LIB='BOOK_DIR'
MODE='I'
LRECL=196
) [1]
```

The values of the attributes, 'LIB', 'MODE' and 'LRECL' are inserted into the list. Although the argument 'DATA' has not been specified, the function recognises that a dataset has been specified because the argument to *source-name* comprises a libname and a dataset name from the specified library.

Example – getting attribute information for a data view

In this example, attributes are returned from a specified data view into a list. The resulting list is then written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
LIBNAME BOOK_DIR 'c:\temp\books';
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'LIB');
  lr1 = SETNITEMC(lid1, '', 'MODE');
  lr1 = SETNITEMN(lid1, ., 'ANOBS');
  lr1 = SETNITEMN(lid1, ., 'INDEX');
  lr1 = SETNITEMN(lid1, ., 'LRECL');
  lr1 = SETNITEMN(lid1, ., 'ISSUBSET');

  dsc = DESCRIBE('book_dir.histbooks', lid1, 'view');

  CALL PUTLIST(lid1, , 0);

RUN;
```

This produces the following output:

```
(LIB='BOOK_DIR'
MODE='I'
ANOBS=1
INDEX=1
LRECL=196
CRDTE=1816773917.3
) [1]
```

The values of the attributes 'LIB', 'MODE', 'ANOBS', 'INDEX', 'LRECL', and 'CRDTE' are inserted into the list. In this case, the data is contained in a data view because *type* is specified as 'VIEW'.

Example – getting attribute information for a catalog entry

In this example, attributes are returned from a specified catalog entry into a list. The resulting list is then written to the log using `CALL PUTLIST` [\(page 1737\)](#).

```
DATA _NULL_;

  lid1 = MAKELIST();

  lr1 = SETNITEMC(lid1, '', 'DESC');
  lr1 = SETNITEMC(lid1, '', 'EDESC');
  lr1 = SETNITEMC(lid1, , 'CRDATE');
  lr1 = SETNITEMN(lid1, ., 'DATE');

  dsc = DESCRIBE('book_dir.bookscat.history1.CATAMS', lid1);

  CALL PUTLIST(lid1,,0);

  gi = GETNITEMN(lid1, 'date');
  PUT gi = date.;

RUN;
```

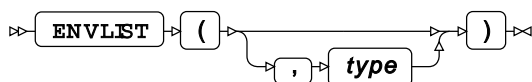
This produces the following output:

```
(DESC='History subset'
 EDESC=' '
 CRDATE='09/11/2017'
 DATE=21073
) [1]
Converted date 11SEP17
```

The value for 'CRDATE' has been returned as a string, while the value for 'DATE' has been returned as the number of days since epoch. In this example, the value for 'DATE' is obtained from the list and written to the log using the `DATE.` format to show that the values of 'CRDATE' and 'DATE' are the same. The filename is recognised as a catalog entry as it comprises four elements.

ENVLIST

Creates a list.



Unlike `MAKELIST` [\(page 1667\)](#) or `MAKENLIST` [\(page 1668\)](#), you do not have to specify the number of items in the list, or the item names in the list. This does mean, however, that you have to be careful how you insert items into the list.

Return type: List

The identifier of the list. Once a list has been created using this function, the same identifier is created each time you use this function. That is, if you use the function twice, the same identifier is returned, not a different identifier each time as would happen with `MAKELIST` or `MAKENLIST`.

Note:

Different identifiers would be returned in different local and global scopes depending on how *type* is set.

type

Optional argument

"G"

Currently has no effect; provided for compatibility.

Because a list created with this function cannot by default grow automatically, you must consider how you insert items into it. For example, if you attempt to use `SETITEMC` [↗](#) (page 1696) to create a list, you must set the *autogrow* argument to `Y` to enable items to be inserted. However, you could use `INSERTC` [↗](#) (page 1654) instead, as this function automatically grows the list.

Example

In this example, a new list is created. A series of named items are entered into the list. The result is written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
DATA _NULL_;

  lid1 = ENVLIST('g');

  lr1 = SETITEMC(lid1, 'bicycle', 1, 'y');
  lr1 = SETITEMC(lid1, 'plane', 2, 'y');
  lr1 = SETITEMC(lid1, 'horse', 3, 'y');

  CALL PUTLIST(lid1,,0);

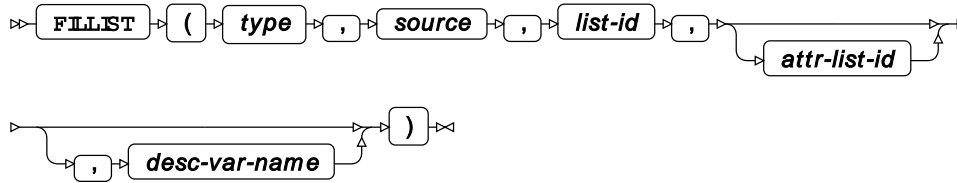
RUN;
```

This produces the following output:

```
('bicycle'
'plane'
'horse'
)[1]
```


FILLIST

Creates a list using the items in a previously saved list.



The list is retrieved from a catalog entry or file created using [SAVELIST](#) (page 1683).

Return type: Numeric

type

Type: Character

Specifies the type of file into which the list was saved. The type can be:

'CATALOG'

The list is retrieved from a catalog entry. Lists can be stored in catalogs as LOG, OUTPUT, SOURCE or SLIST entries. The SLIST catalog entry is provided specifically for storing lists.

'FILE'

The list is retrieved from a file.

'FILEREf'

The list is retrieved from a file identified by a fileref.

The file or catalog entry from which the list is retrieved is specified using *source*.

source

Type: Character

Specifies the filename, fileref or catalog entry name that contains the items you want to insert into a list. The value should correspond to the type of file you specified in *type*. For example, if you are writing the list into a filesystem file, you would need to specify 'FILE'

list-id

Type: List

The identifier for the list.

attr-list-id

Optional argument

Type: List

The identifier of a list that contains a list of attributes corresponding to items in *list-id*. An attribute list defines the appearance of list items when displayed; it is saved with the corresponding list in `SAVELIST`.

Attribute lists are only saved with catalog entry types `LOG`, `OUTPUT` and `SOURCE`.

desc-var-name

Optional argument

Type: Var

Variable into which the list description is returned. When a list is saved with `SAVELIST`, a description can be applied. This argument enables you to obtain the description, if it exists.

For information on list attributes, see `SAVELIST` [↗](#) (page 1683).

Basic example

In this example, a list is created that contains items retrieved from a file that was previously created with `SAVELIST`. The resulting list is then written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
FILENAME listf 'C:\temp\newlist';
DATA _NULL_;
  lid1 = MAKELIST(6, 'g');
  flr = FILLIST('FILEREf', 'listf', lid1);
  CALL PUTLIST(lid1,,0);
RUN;
```

This produces the following output:

```
('bicycle'
'plane'
'horse'
'car'
'boat'
'train'
)[58]
```

Example – retrieving a list and attribute list from a catalog entry

In this example, a list is created that contains three items retrieved from a file that was previously created with `SAVELIST`. The resulting list is then written to the log using `CALL PUTLIST` [↗](#) (page 1737).

```
LIBNAME temp 'C:\temp';
DATA _NULL_;
  lid1 = MAKELIST(3, 'g');
  lid2 = ENVLIST('g');
  flr = FILLIST('CATALOG', 'temp.listtest.newlist.source', lid1, lid2);
  CALL PUTLIST(lid1,,0);
RUN;
```

This produces the following output:

```
('bicycle'  
'plane'  
'horse'  
) [1]
```

Example – retrieving a list and title

In this example, a list is created that contains three items retrieved from a file that was previously created with by using `SAVELIST`. The title is also retrieved. The resulting list and title are then written to the log using `CALL PUTLIST` [\(page 1737\)](#).

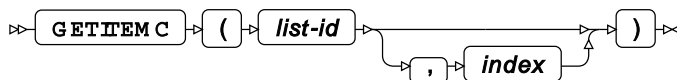
```
LIBNAME temp 'C:\temp';  
DATA _NULL_;  
  FORMAT title $50.;  
  title='';  
  lid1 = MAKELIST(3, 'G');  
  flr = FILLIST('CATALOG', 'temp.listtest.newlist.slist', lid1,, title);  
  CALL PUTLIST(lid1,, 0);  
  PUT title;  
RUN;
```

This produces the following output:

```
('bicycle'  
'plane'  
'horse'  
) [1]  
List of transport modes
```

GETITEMC

Returns the character or string at the specified position in the list.



Return type: Character

The value assigned to the specified item. If the item does not contain a character, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of the character or string. The default is 1.

Example

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMC(lid1, "car", 1);
  rc = SETITEMC(lid1, "train", 2);
  rc = SETITEMC(lid1, "bicycle", 3);
  rc = SETITEMC(lid1, "plane", 4);

  DO y = 1 TO listlen(lid1);
    pc = GETITEMC(lid1, y);
    PUT "List item " y "is: " pc;
  END;

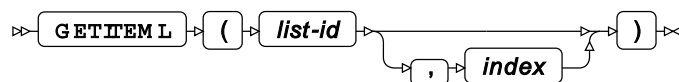
RUN;
```

This produces the following output:

```
List item 1 is: car
List item 2 is: train
List item 3 is: bicycle
List item 4 is: plane
```

GETITEML

Returns the list identifier of the sublist at a specified position in a list.



Return type: List

The list identifier assigned to the specified item. If the item does not contain an identifier, a missing value is returned and an error message written to the log.

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of the sublist. The default is 1. If the first item is not a list identifier, then an error message is returned.

Example

In this example, a list is created using the [MAKENLIST](#) (page 1667) and [SETNITEMC](#) (page 1696) functions. Another list is then created that is then inserted into the first list as a sublist. The list identifier of the sublist is then found. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  lr2 = SETNITEMC(lid2, 'car' , 'lst1');
  lr2 = SETNITEMC(lid2, 'train', 'lst2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'lst3');

  ir = INSERTL(lid1, lid2, 2, 'Sub1');

  gi = GETITEML(lid1, 2);

  PUT 'The identifier of the sublist is: ' ir;

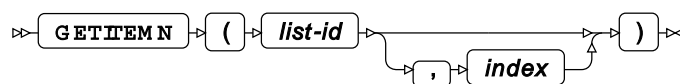
RUN;
```

This produces the following output:

```
The identifier of the sublist is: 2
```

GETITEMN

Returns the number at the specified position in the list.



Return type: Numeric

The number assigned to the specified item. If the named item is assigned anything other than a number, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of the number. The default is 1.

Example

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMN` [\(page 1703\)](#) functions. The items are then extracted from the list using `GETITEMN` [\(page 1631\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 250, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 600, 4);

  DO y = 1 TO listlen(lid1);
    gi = GETITEMN(lid1, y);
    PUT 'List item ' y 'is: ' gi;
  END;

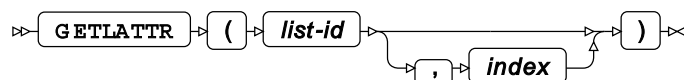
RUN;
```

This produces the following output:

```
List item 1 is: 100
List item 2 is: 250
List item 3 is: 300
List item 4 is: 600
```

GETLATTR

Returns the attributes that are set for a list or for a list item.



Return type: Character

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of an item.

If you specify *index*, then the attributes are returned for an item; if you do not, then the attributes of the list are returned.

Example

In this example, a list is created using the `MAKELIST` [↗](#) (page 1667) and `SETITEMN` [↗](#) (page 1703) functions. The attributes for the list, and for the second item in the list, are returned using `GETLATTR`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  attr4list = GETLATTR(lid1);
  PUT 'Attributes for list: ' attr4list;

  attr4item = GETLATTR(lid1,2);
  PUT 'Attributes for second item: ' attr4item;

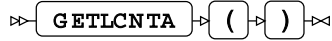
RUN;
```

This produces the following output:

```
Attributes for list:
DELETE UPDATE NOFIXEDTYPE NOFIXEDLENGTH ANYNAMES DUPNAMES NOCHARONLY NONUMONLY COPY
NOHONORCASE
Attributes for second item: ACTIVE WRITE NOAUTO DELETE UPDATE NOFIXEDTYPE
```

GETLCNTA

Returns the number of lists that remain open.



Return type: Numeric

Example

In this example, a loop is used to create three lists and delete one of them, three times; therefore, nine lists are created of which three are deleted. This function is then used to return the number of lists that remain open. The result is written to the log.

```
DATA _NULL_;

DO i = 1 TO 3;

    lid1 = MAKENLIST('l', 'v1', 'v2', 'v3');

    lr = SETNITEMC(lid1, 'car', 'v1');
    lr = SETNITEMC(lid1, 'bicycle', 'v2');
    lr = SETNITEMC(lid1, 'train', 'v3');

    lid2 = MAKENLIST('l', 'v1', 'v2', 'v3');

    PUT 'About to delete id: ' lid1;
    dl = DELLIST(lid1);

    lid3 = MAKENLIST('l', 'var1', 'var2', 'var3');

    lr = SETNITEMC(lid3, 'car', 'var1');
    lr = SETNITEMC(lid3, 'bicycle', 'var2');
    lr = SETNITEMC(lid3, 'train', 'var3');

END;

ra = GETLCNTA();
PUT 'The number of lists that remain open is: ' ra;

RUN;
```

This produces the following output:

```
About to delete id: 1
About to delete id: 4
About to delete id: 7
The number of lists that remain open is: 6
```

Six lists remain open of the nine that were created; these have the identifiers 2, 3, 5, 6, 8 and 9.

GETLCNTP

Returns the total number of lists that resulted from create and delete operations while `CALL LISTPROF` is active.



The number of lists that result from create and delete operations is calculated between a `CALL LISTPROF('ON')` and a `CALL LISTPROF('OFF')`. The number is reset whenever a `CALL LISTPROF('ON')` is used. `CALL LISTPROF('ON')` remains active across DATA steps.

See `CALL LISTPROF` [↗](#) (page 1735) for more information.

Return type: Numeric

If you use `GETLCNTP` without first using `CALL LISTPROF`, 0 is returned.

Example

In this example, two lists are created. One list is deleted, and then another opened. This is repeated three times in a loop. The result is written to the log.

```

DATA _NULL_;

  CALL LISTPROF('on');
  DO i = 1 TO 3;
    lid1 = MAKENLIST('1', 'v1', 'v2', 'v3');
    lid2 = MAKENLIST('1', 'v1', 'v2', 'v3');
    lid3 = MAKENLIST('1', 'var1', 'var2', 'var3');
  END;
  rp = GETLCNTP();
  PUT 'Number of lists created: ' rp;

  rc1 = DELLIST(1);
  rc2 = DELLIST(2);
  rc3 = DELLIST(3);

  ra = GETLCNTP();
  PUT 'Number of lists existing: ' ra;

RUN;

DATA _NULL_;
  lid1 = MAKENLIST('1', 'v1', 'v2', 'v3');
  lid2 = MAKENLIST('1', 'v1', 'v2', 'v3');
  rp = GETLCNTP();
  PUT 'Number of lists created: ' rp;
  rc = dellist(lid2);
  rp = GETLCNTP();
  PUT 'Number of lists existing: ' rp;
  CALL LISTPROF('off');
RUN;

```

The first DATA step, produces the following:

```
Number of lists created: 9
Number of lists existing: 6
```

Because nine lists are created, and three are deleted, six remain.

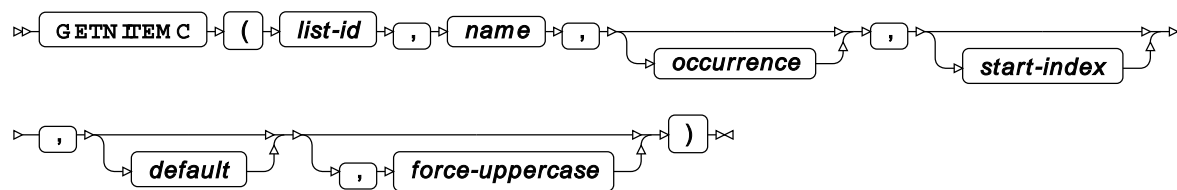
When the second DATA step runs, the output is:

```
Number of lists created: 8
Number of lists existing: 7
```

The total number of lists that have been created is now eight. Two new ones have been added to the six remaining after the previous DATA step. One of the lists was deleted, so the number of lists remaining is seven.

GETNITEMC

Returns the character or string specified for a named list item.



If there is more than one list item with the same name, the value of the first is obtained unless you specify which occurrence to find.

Return type: Character

The value assigned to the specified named item. If the named item does not contain a character, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

name

Type: Character

The name of the item from which you want to obtain a value.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item. The default is 1. This argument is useful when the list contains non-unique item names, and you want to search for an occurrence after an index position.

default

Optional argument

Type: Character

A default value to be used if the named item is not found.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the `SETLATTR` [\(page 1706\)](#) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a list that contains four named items. `SETNITEMC` [↗](#) (page 1712) is then used to set the values of the items. `GETNITEMC` is used to obtain values from specified items. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');

  rx = GETNITEMC(lid1,'var4');
  PUT 'The value is: ' rx;

  rx = GETNITEMC(lid1,'var1');
  PUT 'The value is: ' rx;

RUN;
```

This produces the following output:

```
The value for var4 is: taxi
The value for var1 is: bicycle
```

Example – non-unique item names

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that consists of six named items, where three of the items have non-unique item names. `GETNITEMC` is used to obtain the value of the specified occurrence of a named item. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');
  rc = SETNITEMC(lid1, 'ship', 'var4',2);
  rc = SETNITEMC(lid1, 'car', 'var4',3);

  rx = GETNITEMC(lid1,'var4',2);
  PUT 'The value is: ' rx;

RUN;
```

This produces the following output:

```
The value is: ship
```

Example – getting value of non-unique item name after starting point

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that consists of six named items, where three of the items have non-unique item names. `GETNITEMC` is used to obtain the value of the second occurrence of an item after a specified starting point. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');
  rc = SETNITEMC(lid1, 'ship', 'var4',2);
  rc = SETNITEMC(lid1, 'car', 'var4',3);

  rx = GETNITEMC(lid1,'var4',2,5);
  PUT 'The value is: ' rx;

RUN;
```

This produces the following output:

```
The value is: car
```

The second occurrence after index position five is found.

Example – specifying case

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that holds four named items. The names are specified in upper case. The value for each item is then set using an equivalent lower case name in `SETNITEMC` [↗](#) (page 1712), but with the case forced to upper. The value is then obtained using `GETNITEMC`. The result is written to the log.

```
DATA _NULL_;

  ix = 0;

  lid1 = MAKENLIST('l', 'VAR1','VAR2','VAR3','VAR4');

  mm = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMC(lid1, 'bicycle','var1',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'plane', 'var2',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'horse', 'var3',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4',,,ix,'y');

  CALL PUTLIST(lid1,,0);

  rx = GETNITEMC(lid1,'var1',,,,'roller skate');
  PUT 'Returns: ' rx;

  rx = GETNITEMC(lid1,'VAR4',,,,'roller skate');
  PUT 'Returns: ' rx;

  rx = GETNITEMC(lid1,'var1',,,,'roller skate','y');
  PUT 'Returns: ' rx;

RUN;
```

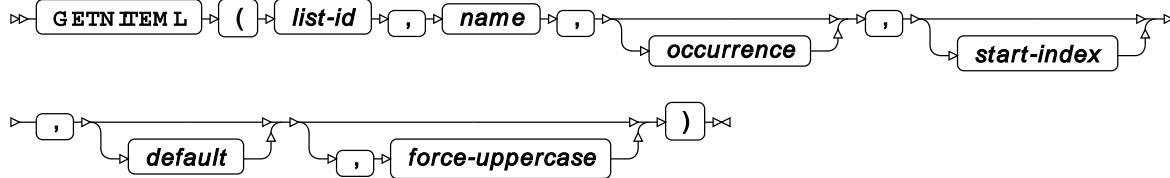
This produces the following output:

```
      (VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
)[1]
Returns: roller skate
Returns: taxi
Returns: bicycle
```

The first use of `GETNITEMC` returns the specified default value `roller skate` because the item name `var1` does not exist (`VAR1` does). The third use of `GETNITEMC` returns the value specified in `SETNITEMC` [↗](#) (page 1668) because the *force-uppercase* argument is set to `Y`.

GETNITEML

Gets the sublist identifier held by the specified named list item.



Returns the list identifier of the sublist, rather than the items in the sublist.

Return type: List

The list identifier assigned to the specified named item. If the named item does not contain an identifier, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

name

Type: Character

The name of the item that holds the sublist identifier.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item that holds the sublist identifier. The default is 1. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

default

Optional argument

Type: List

A default list identifier to be returned if the named item does not exist.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, [MAKENLIST](#) (page 1668) is used to create a new list that contains four named items. A second list is then created in the same way but contains three named items. The items in the lists are set using [SETNITEMC](#) (page 1712) and [SETNITEMN](#) (page 1725). The second list is then set as a sublist on the fourth item in the first list using [SETNITEML](#) (page 1719). [GETNITEML](#) is used to obtain the identifier of the sublist. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  rc = SETNITEMN(lid2, 100, 'ov1');
  rc = SETNITEMN(lid2, 250, 'ov2');
  rc = SETNITEMN(lid2, 310, 'ov3');

  rc = SETNITEML(lid1, lid2, 'var4');

  CALL PUTLIST(lid1,,0);

  rx = GETNITEML(lid1,'var4');
  PUT 'The list identifier is: ' rx;

RUN;
```


This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4=(OV1=100
      OV2=250
      OV3=310
      ) [2]
) [1]
The list identifier is: 2
```

The output shows that the second list was created with the identifier 2; this is the value returned by GETNITEML.

Example – non-unique item names

In this example, `MAKENLIST` [\(page 1668\)](#) is used to create a new list that holds five named items, where two of the items have non-unique item names. A second list is then created in the same way that contains three named items. The items in the lists are set using `SETNITEMC` [\(page 1712\)](#) and `SETNITEMN` [\(page 1725\)](#). The second list is then set as a sublist on the second occurrence of `var4` in the first list using `SETNITEML` [\(page 1719\)](#). `GETNITEML` is used to obtain the identifier of the sublist. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4');

  rc = SETNITEMC(lid1, 'bicycle', 'var1');
  rc = SETNITEMC(lid1, 'plane',   'var2');
  rc = SETNITEMC(lid1, 'horse',   'var3');
  rc = SETNITEMC(lid1, 'taxi',    'var4');
  rc = SETNITEMC(lid1, 'train',   'var4');

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  rc = SETNITEMN(lid2, 100, 'ov1');
  rc = SETNITEMN(lid2, 250, 'ov2');
  rc = SETNITEMN(lid2, 310, 'ov3');

  rc = SETNITEML(lid1, lid2, 'var4',2);

  CALL PUTLIST(lid1,,0);

  rx = GETNITEML(lid1,'var4',2);
  PUT 'The list identifier is: ' rx;

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='train'
VAR4=(OV1=100
      OV2=250
      OV3=310
    ) [6]
)[5]The list identifier is: 6
```

The output shows that the second list was created with the identifier 6; this is the value returned by GETNITEML.

Example – getting value of non-unique item name after starting point

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that holds six named items, where three of the items have non-unique item names. A second list is then created in the same way that contains three named items. The items in the lists are set using `SETNITEMC` [↗](#) (page 1712) and `SETNITEMN` [↗](#) (page 1725). `SETNITEML` [↗](#) (page 1719) is then used to set the second list as a sublist on the second occurrence of `var4` after a specified starting point. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');

  lr1 = SETNITEMC(lid1, 'bicycle', 'var1');
  lr1 = SETNITEMC(lid1, 'plane',   'var2');
  lr1 = SETNITEMC(lid1, 'horse',   'var3');
  lr1 = SETNITEMC(lid1, 'taxi',    'var4');
  lr1 = SETNITEMC(lid1, 'train',   'var4',2);
  lr1 = SETNITEMC(lid1, 'bus',     'var4',3);

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  lr2 = SETNITEMN(lid2, 100, 'ov1');
  lr2 = SETNITEMN(lid2, 250, 'ov2');
  lr2 = SETNITEMN(lid2, 310, 'ov3');

  si = SETNITEML(lid1, lid2, 'var4',3);

  CALL PUTLIST(lid1,,0);

  rx = GETNITEML(lid1,'var4',2,5);
  PUT 'The list identifier is: ' rx;

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
VAR4='train'
VAR4=(OV1=100
      OV2=250
      OV3=310
    ) [4]
) [3]
The list identifier is: 4
```

The second occurrence after index position five is found. The output shows that the second list was created with the identifier 6; this is the value returned by `GETNITEML`.

Example – specifying case

In this example, `MAKENLIST` [\(page 1668\)](#) is used to create a new list that holds four named items, specified in uppercase. A second list is then created in the same way that contains three named items. The items in the lists are set using `SETNITEMC` [\(page 1712\)](#) and `SETNITEMN` [\(page 1725\)](#). The second list is then set as a sublist on the item `VAR4` in the first list using `SETNITEML` [\(page 1719\)](#).

Because the 'HONORCASE' attribute has been set for the first list (using `SETLATTR` [\(page 1706\)](#)), references to the items in that list must use the same case as that specified when the list was created. Although the `SETNITEML` statement has `var4` specified in lowercase, the option has been set to force it to uppercase. The list will, therefore, be correctly assigned to `VAR4`.

Finally, `GETNITEML` is used in various formats to obtain the identifier of the sublist. The result is written to the log.

```
DATA _NULL_;

  ix = 0;

  lid1 = MAKENLIST('l', 'VAR1','VAR2','VAR3','VAR4');

  lr1 = SETNITEMC(lid1, 'bicycle', 'VAR1');
  lr1 = SETNITEMC(lid1, 'plane',   'VAR2');
  lr1 = SETNITEMC(lid1, 'horse',   'VAR3');
  lr1 = SETNITEMC(lid1, 'taxi',    'VAR4');

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  lr2 = SETNITEMN(lid2, 100, 'ov1');
  lr2 = SETNITEMN(lid2, 250, 'ov2');
  lr2 = SETNITEMN(lid2, 310, 'ov3');

  mm = SETLATTR(lid1, 'HONORCASE');

  si = SETNITEML(lid1, lid2, 'var4',,,ix,'y');

  CALL PUTLIST(lid1,,0);

  gi = GETNITEML(lid1,'var1',,,0);
  PUT 'Returns: ' gi;
```

```

gi = GETNITEML(lid1,'VAR2',,,0,'Y');
PUT 'Returns: ' gi;

gi = GETNITEML(lid1,'var4',,,0,'Y');
PUT 'Returns: ' gi;

RUN;

```

This produces the following output:

```

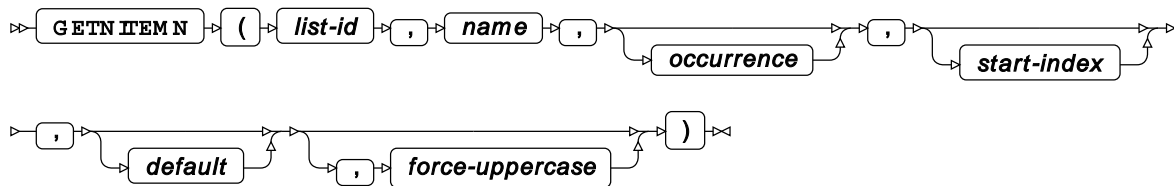
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4=(OV1=100
      OV2=250
      OV3=310
      ) [2]
) [1]
Returns: 0
NOTE: Argument to function GETNITEML at line 1218 column 8 is invalid
Returns: .
Returns: 2

```

The first use of GETNITEML returns 0 because the item name `var1` does not exist (`VAR1` does). The second use of GETNITEML returns an error message and a missing value, as the identifier `VAR2` exists but no list is assigned to it. The third use of GETNITEML returns 2, the identifier of the sublist at the item name specified by SETNITEML, because GETNITEML has the *force-uppercase* argument set to `Y`.

GETNITEMN

Returns the number specified for a named list item.



If there is more than one occurrence of the same named item, the value of the first is obtained, unless you specify which occurrence to find.

Return type: Numeric

The number assigned to the specified named item. If the named item does not contain a number, a missing value is returned and an error message is written to the log.

list-id

Type: List

The identifier for the list.

name

Type: Character

The name of the item from which you want to obtain a value.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

The ordinal position in the list at which to start searching for the named item. By default, the search starts at the first item. This argument is useful when the list contains non-unique item names, and you want to search for an occurrence after an index position.

default

Optional argument

Type: Numeric

A default value to be used if the named item is not found.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, a new list is created that contains four named items. `GETNITEMN` is used to obtain values from specified items. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  rx = GETNITEMC(lid1,'var4');
  PUT 'The value of var4 is: ' rx;

  rx = GETNITEMC(lid1,'var1');
  PUT 'The value of var1 is: ' rx;

RUN;
```

This produces the following output:

```
The value of var4 is: 102
The value of var1 is: 100
```

Example – non-unique item names

In this example, a new list is created that consists of six named items, where three of the items have non-unique item names. `GETNITEMN` is used to obtain values from specified items. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');

  lr1 = SETNITEMN(lid1, 100, 'var1');
  lr1 = SETNITEMN(lid1, 250, 'var2');
  lr1 = SETNITEMN(lid1, 600, 'var3');
  lr1 = SETNITEMN(lid1, 800, 'var4');
  lr1 = SETNITEMN(lid1, 60, 'var4',2);
  lr1 = SETNITEMN(lid1, 20, 'var4',3);

  rx = GETNITEMN(lid1,'var4',2);
  PUT 'The value is: ' rx;

RUN;
```

This produces the following output:

```
The value is: 60
```

Example – getting value of non-unique item name after starting point

In this example, a new list is created that consists of six named items, where three of the items have non-unique item names. `GETNITEMN` is used to obtain values from the second occurrence of an item after a specified start point. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');

  lr1 = SETNITEMN(lid1, 100, 'var1');
  lr1 = SETNITEMN(lid1, 250, 'var2');
  lr1 = SETNITEMN(lid1, 600, 'var3');
  lr1 = SETNITEMN(lid1, 800, 'var4');
  lr1 = SETNITEMN(lid1, 60, 'var4',2);
  lr1 = SETNITEMN(lid1, 20, 'var4',3);

  rx = GETNITEMN(lid1,'var4',2,5);
  PUT 'The value is: ' rx;

RUN;
```

This produces the following output:

```
The value is: 20
```

The second occurrence after index position five is found.

Example – specifying case

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that consists of four named items. The names are specified in upper case. The value for each item is then set using an equivalent lower case name in `SETNITEMN` [↗](#) (page 1725). The value is then obtained using `GETNITEMN`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4');

  mm = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMN(lid1, 100, 'var1',,,ix,'y');
  lr1 = SETNITEMN(lid1, 250, 'var2',,,ix,'y');
  lr1 = SETNITEMN(lid1, 310, 'var3',,,ix,'y');
  lr1 = SETNITEMN(lid1, 40,  'var4',,,ix,'y');

  CALL PUTLIST(lid1,,0);

  gi = GETNITEMN(lid1,'var1',,, 0);
  PUT 'Returns: ' gi;

  gi = GETNITEMN(lid1,'VAR4',,, 0, 'y');
  PUT 'Returns: ' gi;

  gi = GETNITEMN(lid1,'var1',,, 0, 'y');
  PUT 'Returns: ' gi;

RUN;
```

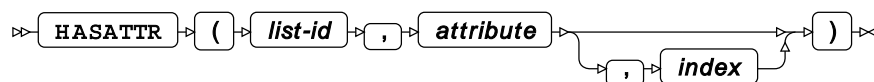
This produces the following output:

```
(VAR1=100
VAR2=250
VAR3=310
VAR4=40
)[1]Returns: 0
Returns: 40
Returns: 100
```

The first use of `GETNITEMN` returns the specified default value 0 because the item name `var1` does not exist (`VAR1` does). The third use of `GETNITEMN` returns the value specified in `SETNITEMN` because the *force-uppercase* argument is set to `Y`.

HASATTR

Returns a value indicating whether a list or list item has a specified attribute value.



Return type: Numeric

0 (zero) if the attribute is not assigned, 1 if it is.

list-id

Type: List

The identifier for the list.

attribute

An attribute that the list or list item might have. The **G** and **L** attributes are specified when a list is created; for all other attributes, see [SETLATTR](#) [↗](#) (page 1706).

"G"

Specified when a list is created. Applies to lists only.

"L"

Specified when a list is created. Applies to lists only.

"ACTIVE"

Applies to list items only.

"ANYNAMES"

Applies to lists only.

"AUTO"

Applies to list items only.

"CHARONLY"

Applies to lists only.

"COPY"

Applies to lists only.

"DELETE"

Applies to lists and to list items.

"DUPNAMES"

Applies to lists only.

"FIXEDLENGTH"

Applies to lists only.

"FIXEDTYPE"

Applies to lists and to list items.

"HONORCASE"

Applies to lists only.

"NOHONORCASE"

Applies to lists only.

"INACTIVE"

Applies to list items only.

"NOAUTO"

Applies to list items only.

"NOCHARONLY"

Applies to lists only.

"NOCOPY"

Applies to lists only.

"NODELETE"

Applies to lists and to list items.

"NODUPNAMES"

Applies to lists only.

"NOFIXEDLENGTH"

Applies to lists only.

"NOFIXEDTYPE"

Applies to lists and to list items.

"NONUMONLY"

Applies to lists only.

"NOUPDATE"

Applies to lists and to list items.

"NOWRITE"

Applies to list items only.

"NUMONLY"

Applies to lists only.

"SASNAMES"

Applies to lists only.

"UPDATE"

Applies to lists and to list items.

"WRITE"

Applies to list items only.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of an item.

If you do not specify *index* when *attribute* can apply both to a list and to a list item, the function returns the status of the attribute for the list. If you specify *index* when *attribute* only applies to the list, a message is returned in the log. If you do not specify *index* when *attribute* only applies to list items, a message is returned in the log.

Example

In this example, a list is created using the `MAKENLIST` [\(page 1668\)](#) and `SETNITEMC` [\(page 1696\)](#) functions. Specified attributes are then checked to see if they have been set (the attributes are those set by default in this example) . The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  hao = IFC(HASATTR(lid1,'fixedlength'),'The list has the attribute',
            'The list does not have the attribute');
  PUT hao 'fixedlength';

  hao = IFC(HASATTR(lid1,'nofixedlength'),'The list has the attribute',
            'The list does not have the attribute');
  PUT hao 'nofixedlength';

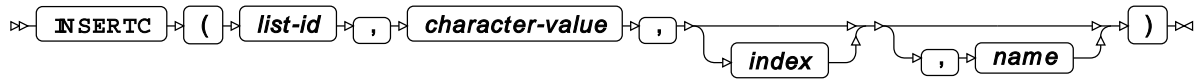
RUN;
```

This produces the following output:

```
The list does not have the attribute fixedlength
The list has the attribute nofixedlength
```

INSERTC

Inserts a character or string at a specified position in the list.



The character or string is inserted at the position specified, and all other list items are shifted to the right. If no position is specified, the item is inserted at the first position in the list. The list automatically grows to accommodate the new item.

Return type: Numeric

list-id

Type: List

The identifier for the list.

character-value

Type: Character

The character or string to insert.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which the character or string is inserted. By default, this is 1.

name

Optional argument

Type: Character

If the list contains named items, a name for the item.

If you want to append the item to the end of the list, you can specify a value for *index* that is one greater than the length of the list. For example, if the list is four items long, and you want to add an item at the end of the list, you would specify the value 5 for *index*. If you specify a number greater than the length of the list plus 1, however, an error message is returned.

Basic example

In this example, a new string item is inserted into a list at the third position. The items are then extracted from the list using `GETITEMC` [↗](#) (page 1629). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMC(lid1, "car", 1);
  rc = SETITEMC(lid1, "train", 2);
  rc = SETITEMC(lid1, "bicycle", 3);
  rc = SETITEMC(lid1, "plane", 4);

  rc = INSERTC(lid1, "ship", 3);

  ll = LISTLEN(lid1);
  PUT "The length of the list is: " ll;

  DO y = 1 TO listlen(lid1);
    pc = GETITEMC(lid1, y);
    PUT "List item " y "is: " pc;
  END;

RUN;
```

This produces the following output:

```
The length of the list is: 5
List item 1 is: car
List item 2 is: train
List item 3 is: ship
List item 4 is: bicycle
List item 5 is: plane
```

The new item has been placed at the third position in the list, and the length of the list has grown to five items.

Example – inserting a named variable at the end of the list

In this example, a new named string item is inserted at the end of a list of named items. The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST("g", "var1", "var2", "var3", "var4");

  rc = SETNITEMC(lid1, "car", "var1");
  rc = SETNITEMC(lid1, "train", "var2");
  rc = SETNITEMC(lid1, "bicycle", "var3");
  rc = SETNITEMC(lid1, "plane", "var4");

  rc = INSERTC(lid1, "ship", 5, "new_var");

  ll = LISTLEN(lid1);
  PUT "The length of the list is: " ll;

  nv = GETNITEMC(lid1, "new_var");
  PUT "The new item is: " nv;

  DO y = 1 TO listlen(lid1);
    pc = GETITEMC(lid1, y);
    PUT "List item " y "is: " pc;
  END;

RUN;
```

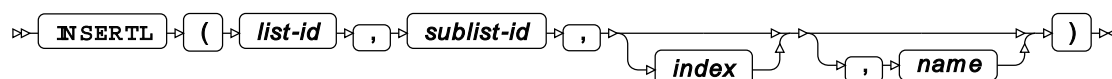
This produces the following output:

```
The length of the list is: 5
The new item is: ship
List item 1 is: car
List item 2 is: train
List item 3 is: bicycle
List item 4 is: plane
List item 5 is: ship
```

The new item has been placed at the end of the list, and the length of the list has grown to five items. `GETNITEMN` [\(page 1646\)](#) has been used to get the item from the list using the item name supplied in `INSERTN`.

INSERTL

Insert a sublist into the specified list.



The sublist is inserted at the first position in the list, unless specified otherwise. When a sublist is inserted into a list, all other list items are moved to the right by one position. For example, if you insert a sublist at position two, the sublist occupies position two and the item that was at that position is moved to position three.

The sublist itself is not inserted into the list; instead, a pointer to the corresponding list identifier is inserted.

Return type: Numeric

The identifier of the list into which the sublist is inserted.

list-id

Type: List

The identifier for the list.

This is the list into which sublist identified by *sublist-id* will be inserted.

sublist-id

Type: List

The identifier of the list that is to be inserted into *list-id*.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which you want to insert the sublist. By default this is 1.

name

Optional argument

Type: Character

A name for the sublist.

Basic example

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMC](#) (page 1696) functions. A second list is similarly created, and then inserted in the first list using this function. The list and sublist are displayed in the log using [CALL PUTLIST](#) (page 1737).

```
DATA _NULL_;

  lid1 = MAKENLIST("g", "var1", "var2", "var3", "var4");

  rc = SETNITEMC(lid1, "ship", "var1");
  rc = SETNITEMC(lid1, "plane", "var2");
  rc = SETNITEMC(lid1, "horse", "var3");
  rc = SETNITEMC(lid1, "taxi", "var4");

  lid2 = MAKENLIST("g", "lst1", "lst2", "lst3");

  rc = SETNITEMC(lid2, "car", "lst1");
  rc = SETNITEMC(lid2, "train", "lst2");
  rc = SETNITEMC(lid2, "bicycle", "lst3");

  rc=INSERTL(lid1,lid2,2);

  PUT "The identifier: " rc;

  CALL PUTLIST(lid1);

RUN;
```

This produces the following output:

```
The identifier: 1
(VAR1='ship' (LST1='car' LST2='train' LST3='bicycle' ) [2] VAR2='plane' VAR3='horse'
  VAR4='taxi'
) [1]
```


Example – getting a sublist after insertion

In this example, a list is created using the `MAKENLIST` (page 1668) and `SETITEMC` (page 1696) functions. A second list is similarly created, and then inserted in the first list using this function. `GETITEML` (page 1630) is then used to get the sublist. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST("g", "var1", "var2", "var3", "var4");

  rc = SETNITEMC(lid1, "ship", "var1");
  rc = SETNITEMC(lid1, "plane", "var2");
  rc = SETNITEMC(lid1, "horse", "var3");
  rc = SETNITEMC(lid1, "taxi", "var4");

  lid2 = MAKENLIST("g", "lst1", "lst2", "lst3");

  rc = SETNITEMC(lid2, "car", "lst1");
  rc = SETNITEMC(lid2, "train", "lst2");
  rc = SETNITEMC(lid2, "bicycle", "lst3");

  rc=INSERTL(lid1,lid2,2);
  PUT "The identifier of the list: " rc;

  rc = GETITEML(lid1, 2);
  PUT "The identifier of the sublist: " rc;

RUN;
```

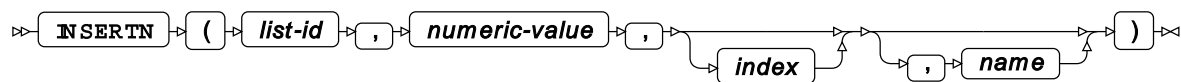
This produces the following output:

```
The identifier of the list: 1
The identifier of the sublist: 2
```

This shows that the sublist into a list as an identifier, and is also retrieved as an identifier.

INSERTN

Inserts a number at a specified position in a list.



The number is inserted at the position specified, and all other list items are shifted to the right. If no position is specified, the item is inserted at the first position in the list. The list automatically grows to accommodate the new item.

Return type: Numeric

list-id**Type:** List

The identifier for the list.

numeric-value**Type:** Numeric

The value to be inserted into the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which the number is inserted. By default this is 1.

name

Optional argument

Type: Character

If the list contains named items, a name for the item.

If you want to append the item to the end of the list, you can specify a value for *index* that is one greater than the length of the list. For example, if the list is four items long, and you want to add an item at the end of the list, you would specify the value 5 for *index*. If you specify a number greater than the length of the list plus 1, however, an error message is returned.

Basic example

In this example, a new numeric item is inserted into a list at the third position. The items are then extracted from the list using `GETITEMN` [↗](#) (page 1631). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST(4);

  rc = SETITEMN(lid1, 100, 1);
  rc = SETITEMN(lid1, 200, 2);
  rc = SETITEMN(lid1, 300, 3);
  rc = SETITEMN(lid1, 400, 4);

  rc = INSERTN(lid1, 450, 3);

  len = LISTLEN(lid1);

  PUT 'The length of the list is: ' len;

  DO y = 1 TO LISTLEN(lid1);
    pc = GETITEMN(lid1);
    PUT 'List item ' y 'is: ' pc;
  END;

RUN;
```

This produces the following output:

```
The length of the list is: 5
List item 1 is: 100
List item 2 is: 200
List item 3 is: 450
List item 4 is: 300
List item 5 is: 400
```

The new item has been placed at the third position in the list, and the length of the list has grown to five items.

Example – inserting a named variable at the end of the list

In this example, a new named numeric item is inserted at the end of a list of named items. The items are then extracted from the list using `GETNITEMN` (page 1631). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMN(lid1, 100, 'var1');
  lr1 = SETNITEMN(lid1, 200, 'var2');
  lr1 = SETNITEMN(lid1, 300, 'var3');
  lr1 = SETNITEMN(lid1, 400, 'var4');

  isn = INSERTN(lid1, 450, 5, 'new_var');

  len = LISTLEN(lid1);
  PUT 'The length of the list is: ' len;

  gi = GETNITEMN(lid1, 'new_var');
  PUT 'The new item is: ' rc;

  DO y = 1 TO LISTLEN(lid1);
    gin = GETNITEMN(lid1, y);
    PUT 'List item ' y 'is: ' gin;
  END;

RUN;
```

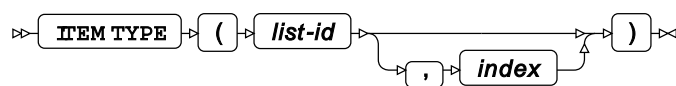
This produces the following output:

```
The length of the list is: 5
The new item is: 450
List item 1 is: 100
List item 2 is: 200
List item 3 is: 300
List item 4 is: 400
List item 5 is: 450
```

The new item has been placed at the end of the list, and the length of the list has grown to five items. `GETNITEMN` (page 1646) has been used to get the item from the list using the item name supplied in `INSERTN`.

ITEMTYPE

Return the type of the item at a specified position in a list.



Return type: Character

The type can be C (character), N (numeric) or L (list).

list-id**Type:** List

The identifier for the list.

index

Optional argument

Type: Numeric

The ordinal position of the item in the list. By default, this is the first item.

Basic example

In this example, the type of the third item in the list is returned. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMC(lid1, 'car', 1);
  rc = SETITEMN(lid1, 200, 2);
  rc = SETITEMN(lid1, 300, 3);
  rc = SETITEMN(lid1, 400, 4);

  pc = ITEMTYPE(lid1);
  PUT 'The type of the item is: ' pc;

  pc = ITEMTYPE(lid1, 3);
  PUT 'The type of the item is: ' pc;

RUN;
```

This produces the following output:

```
The type of the item is: C
The type of the item is: N
```

In the first use of `ITEMTYPE`, no index position is specified, so the type of the value at the first list position (the default) is returned.

Example – using type to choose function to use to get items

In this example, the type of each item is checked, and the appropriate function is then used to get that item. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 400, 4);

  DO i = 1 TO 4;

    IF itemtype(lid1,i) = 'C' THEN gic=getitemc(lid1, i);
    IF itemtype(lid1,i) = 'C' THEN PUT gic;
    IF itemtype(lid1,i) = 'N' THEN gin=getitemn(lid1, i);
    IF itemtype(lid1,i) = 'N' THEN PUT gin;

  END;

RUN;
```

This produces the following output:

```
car
200
300
400
```

In the first use of `ITEMTYPE`, no index position is specified, so the type of the value at the first list position (the default) is returned.

LISTLEN

Returns the number of items in a list.

➤ **LISTLEN** ➤ (➤ *list-id* ➤) ➤

Return type: Numeric

list-id

Type: List

The identifier for the list.

Example

In this example, the length of a list is returned. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 400, 4);

  l1 = LISTLEN(lid1);

  PUT 'The length of the list is: ' l1;

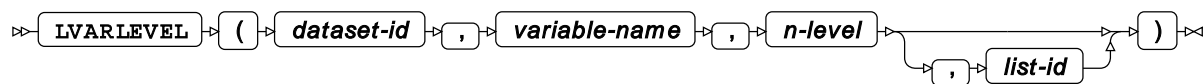
RUN;
```

This produces the following output:

```
The length of the list is: 4
```

LVARLEVEL

Creates a list using the values of a variable in a specified dataset.



You can use this function to create a list from the values for a variable in a dataset; that is, the values in one column.

If there are multiple instances of the same value for a variable, only one instance is placed in the list.

Return type: Numeric

dataset-id

Type: Numeric

The identifier of a dataset opened using the `OPEN` [\(page 752\)](#) function.

variable-name

Type: Character

The name of the variable in the dataset for which values will be converted into list items.

n-level

Type: Var

The name of a variable, into which the number of unique values of *variable-name* is returned.

list-id

Optional argument

Type: List

The identifier of a list that has been created with one of the list creation functions ([MAKELIST](#) (page 1667), [MAKENLIST](#) (page 1668) or [ENVLIST](#) (page 1625)).

If this argument is not specified, a note is written to the log warning that this argument is invalid, unless [CURLIST](#) (page 1607) has previously been used to store a list identifier in memory.

If *list-id* is omitted, the number of unique data items for the specified variable is returned in *n-level*.

The order of the items in the list is the reverse of that in the dataset or input file. For example, if the specified variable comprised three items in the dataset:

```
car
train
bus
```

the resulting items in the list are:

```
bus
train
car
```

Basic example

In this example, a new list is created using the values for the `AUTHOR` variable in the specified dataset, which contains information about books. The result is written to the log.

```
LIBNAME lbooks 'c:\temp\books';
DATA _NULL_;

var = 0;
did = OPEN('lbooks.books');

arc = ATTRN(did, 'nlobsf');
PUT 'The dataset contains ' rc 'observations ';
PUT;

lid = ENVLIST('g');

lvl = LVARLEVEL(did, 'Author', var, lid);

PUT 'The list has ' var 'items: ';
PUT;

DO i = 1 TO var;
    gi = GETITEMC(lid, i);
    PUT gi;
END;

RUN;
```


This produces the following output:

```
The dataset contains 4 observations

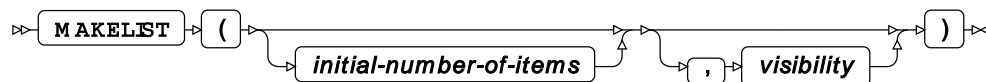
The list has 3 items:

Marwick, Arthur
Hindle, Paul
Hibbert, Christopher
```

The dataset has two observations where the variable `Author` has the same name (Marwick, Arthur). Only one of these is written to the list. The list is in reverse order to the order of the variables in the dataset.

MAKELIST

Creates a list in which items are specified by ordinal position.



The list identifier is an integer. Each time you create a list using this function or the `MAKENLIST` [\(page 1668\)](#) function, a new list identifier is created by incrementing the new identifier by one over the most recently created identifier. For example, if the last list identifier created was 2, a new list will have the identifier 3. These identifiers remain attached to the corresponding list until the WPS server is restarted or the list is cleared.

Return type: List

The list identifier. This is an integer greater than 0 (zero).

initial-number-of-items

Optional argument

Type: Numeric

The number of items in the list.

visibility

Optional argument

Currently has no effect; provided for compatibility.

"G"

This function enables you to create lists in which an item can be located by its ordinal position in the list. If you want to create a list in which an item can be located by a variable name, use `MAKENLIST` [\(page 1667\)](#).

Example

In this example, a new list is created that can hold four items. The result is written to the log.

```
DATA _NULL_;  
  
    lid = MAKELIST(4);  
    PUT 'The list identifier: ' lid;  
  
RUN;
```

This produces the following output:

```
The list identifier: 1
```

If you run the same DATA step again, it returns:

```
The list identifier: 2
```

MAKENLIST

Creates a list in which items are specified by variable name.



The list is a placeholder for a one or more items identified by names. The function returns an identifier that can be used by other list functions. You can enter information into the list, and subsequently get information from the list, using other list functions. These functions use the list identifier specified by this function.

The list identifier is an integer. Each time you create a list using this function or the [MAKELIST](#) (page 1667) function, a new list identifier is created by incrementing the new identifier by one over the most recently created identifier. For example, if the last list identifier created was 2, a new list will have the identifier 3. These identifiers remain attached to the corresponding list until the WPS server is restarted or the list is cleared.

Return type: List

The list identifier. This is an integer greater than 0 (zero).

visibility

"G"

Currently has no effect; provided for compatibility.

item-name-list

Type: Character

One or more variable names for an item in the list, separated by commas; for example 'var1', 'var2', 'var3'.

This function enables you to create lists in which an item can be identified by a variable name. If you want to create a list in which an item can be located by its ordinal position in the list, use `MAKELIST` [\(page 1667\)](#).

Because the identifier for a list is unique, and because variables you specify are associated with an identified list, different lists can contain the same variable names. These variables can then have different values. The variables, and their values, will be unique because they are associated with a specific list.

Example

In this example, a new list is created that can hold four items. The result is written to the log.

```
DATA _NULL_;  
  
    lid = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4',);  
    PUT 'The list identifier: ' lid;  
  
RUN;
```

This produces the following output:

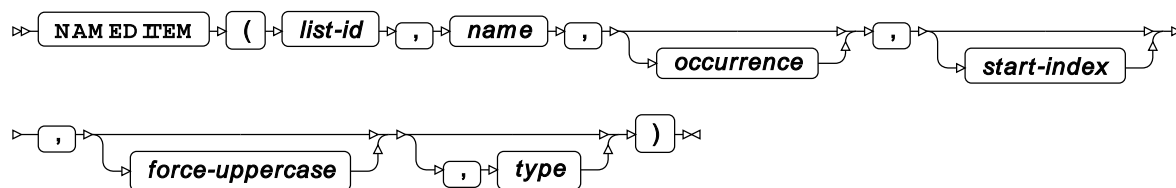
```
The list identifier: 1
```

If you run the same DATA step again, it returns:

```
The list identifier: 2
```

NAMEDITEM

Returns the position of a named item in a list.



Return type: Numeric

The position of the item.

list-id

Type: List

The identifier for the list.

name

Type: Character

The name of the item for which you want the position.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item to set. This must be greater than 0 (zero). The default is 1. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

type

Optional argument

Type: Var

A variable into which the type of value for the named item is returned. If the value is a number, N is returned to the variable; if the value is a character or string, C is returned.

Basic example

In this example, a new list is created that contains four named items. The name of the fourth item is then found. The result is written to the log.

```
DATA _NULL_;

  z='var4';

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4' );

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMN(lid1, 1, 'var4');

  pos = NAMEDITEM(lid1, z);

  PUT 'The variable ' z 'is at position: ' pos;

RUN;
```

This produces the following output:

```
The variable var4 is at position: 4
```

Example – returning information for specified occurrence of named item

In this example, a new list is created that contains six named items, where three of them have the same name. The position of the second occurrence after the third index position is returned. The type of the value associated with the occurrence is also returned. The result is written to the log.

```
DATA _NULL_;

  ty='';

  lid1 = MAKENLIST('l', 'mass_t','class', 'indiv_t', 'class', 'sub_t', 'class' );

  rc = SETNITEMC(lid1, 'coach', 'mass_t');
  rc = SETNITEMN(lid1, 1, 'class');
  rc = SETNITEMC(lid1, 'horse', 'indiv_t');
  rc = SETNITEMN(lid1, 1, 'class',2);
  rc = SETNITEMC(lid1, 'train', 'sub_t');
  rc = SETNITEMN(lid1, 2, 'class',3);

  pos = NAMEDITEM(lid1, 'class',2,2,,ty);

  PUT 'The variable is at position ' pos 'and is of type ' ty;

RUN;
```

This produces the following output:

```
The variable is at position 4 and is of type N
```

Example – specifying case

In this example, a new list is created that contains four named items. The position of the occurrence with and without the specified case is returned. The result is written to the log.

```
DATA _NULL_;

  ip=0;

  lid1 = MAKENLIST('l', 'VAR1','VAR2', 'var3', 'var4' );

  mm = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMC(lid1, 'bicycle','var1',,,ip,'Y');
  lr1 = SETNITEMC(lid1, 'plane', 'var2',,,ip,'Y');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMN(lid1, 1, 'var4');

  pos = NAMEDITEM(lid1, 'var2');
  PUT 'The list item is at position ' pos;

  pos = NAMEDITEM(lid1, 'var2',,, 'Y');
  PUT 'The list item is at position ' pos;

RUN;
```

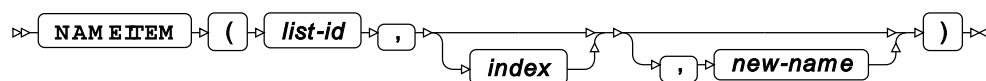
This produces the following output:

```
The list item is at position 0
The list item is at position 2
```

The first use of `NAMEDITEM` returns 0, as there is no item with the name `var2`. The second use of `NAMEDITEM` returns 2, as there is an item with the name `VAR2` at this position, and the *force-uppercase* argument has been set to `Y` to force the case of the specified item name to upper case.

NAMEITEM

Name or rename an item, or return the item name.



For lists that contain named list items, this function can be used to rename an item, or to return the name of an item at a specified position in the list. For lists that contain indexed items, this function can be used to name an item.

Return type: Character

Either the current name of the item at the specified position, or, if you are changing the name, the name before it is changed. The name returned is in upper case characters.

list-id**Type:** List

The identifier for the list.

index

Optional argument

Type: Numeric

The position in the list of the item. The default is 1.

new-name

Optional argument

Type: CharacterThe new name for the item specified by *index*.To return the name of the item without changing it, omit *new-name*.**Example – return name of specified item**

In this example, a list is created using the [MAKENLIST](#) (page 1668) and [SETNITEMC](#) (page 1696) functions. The name of the second item in the list is returned. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3');

  rc = SETNITEMC(lid1, 'car', 'var1');
  rc = SETNITEMC(lid1, 'bicycle', 'var2');
  rc = SETNITEMC(lid1, 'train', 'var3');

  vn = NAMEITEM(lid1, 2);

  PUT 'The item name is: ' vn;

RUN;
```

This produces the following output:

```
The item name is: VAR2
```

Note:

Whether the item name is returned in upper or lower case depends on the functions that previously used the name.

Example – change name of specified item

In this example, a list is created using the `MAKENLIST` [\(page 1668\)](#) and `SETITEMC` [\(page 1696\)](#) functions. The name of the second item in the list is changed. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3');

  rc = SETNITEMC(lid1, 'car', 'var1');
  rc = SETNITEMC(lid1, 'bicycle', 'var2');
  rc = SETNITEMC(lid1, 'train', 'var3');

  cv = NAMEITEM(lid1, 2, 'tw_trans');

  PUT 'The item name changed is: ' cv;

  gin = NAMEITEM(lid1, 2);
  giv = GETNITEMC(lid1, gin);
  PUT 'The item name is now ' gin 'and its value is: ' giv;

RUN;
```

This produces the following output:

```
The item name changed is: VAR2
The item name is now tw_trans and its value is: bicycle
```

Note:

Whether the item name is returned in upper or lower case depends on the functions that previously used the name.

In this example, the name of item `var2` is changed to `tw_trans`. The function returns the value before it is changed. The new name is then used in a `GETNITEMC` function to return the value for that item. This value is same as that to which `var2` was first set.

Example – specify names for indexed items

In this example, a list is created using the `MAKENLIST` [\(page 1668\)](#) and `SETNITEMC` [\(page 1696\)](#) functions. The list items are then named using `NAMEITEM`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3');

  lr1 = SETNITEMC(lid1, 'car', 'var1');
  lr1 = SETNITEMC(lid1, 'bicycle', 'var2');
  lr1 = SETNITEMC(lid1, 'train', 'var3');

  cv = NAMEITEM(lid1, 2, 'tw_trans');

  PUT 'The item name is: ' cv;

  gi = GETNITEMC(lid1, 'tw_trans');
  PUT 'The value of tw_trans is: ' gi;

RUN;
```

This produces the following output:

```
The item name is: VAR2
The value of tw_trans is: bicycle
```

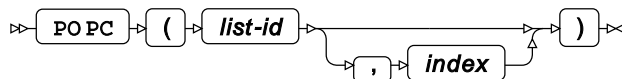
Note:

Whether the item name is returned in upper or lower case depends on the functions that previously used the name.

In this example, the name of item `var2` is changed to `tw_trans`. The function returns the value before it is changed. The new name is then used in a `GETNITEMC` function to return the value for that item. This value is same as that to which `var2` was first set.

POPC

Returns and removes a character or string item from a specified list.



The function returns the value of a specified item in a list, and removes that item from the list. This is known as popping, and such an item has been popped from the list. The function can return the next string item, or a specified string item from a list, depending on the options you specify.

Return type: Character

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list from which to pop the item. The default is 1. If you specify a value greater than the number of items in the list, an error is returned.

If you try to pop an item that is not a character or string, an error message is returned.

Note:

Because popped items are removed from the list, you should be careful when using a loop to set the index for the item to be popped.

Basic example

In this example, all items are extracted from a list using the `POPC` [\(page 1675\)](#) function. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'train', 2);
  lr1 = SETITEMC(lid1, 'bicycle', 3);
  lr1 = SETITEMC(lid1, 'plane', 4);

  DO y = 1 TO 4;
    pc = POPC(lid1);
    PUT 'Item ' y 'in the list was: ' pc;
  END;

  ll = LISTLEN(lid1);
  PUT 'The list length is now: ' ll;

RUN;
```

This produces the following output:

```
List item 1 is: car
List item 2 is: train
List item 3 is: bicycle
List item 4 is: plane
The list length is now: 0
```

The list length is 0 (zero) as all items have been popped from the list.

Example – specifying an ordinal position

In this example, an item at a specified position in the list is extracted using the `POPC` [\(page 1675\)](#) function. The result is written to the log.

```
DATA _NULL_;

pos = 3;

lid1 = MAKELIST(4);

lr1 = SETITEMC(lid1, 'car', 1);
lr1 = SETITEMC(lid1, 'train', 2);
lr1 = SETITEMC(lid1, 'bicycle', 3);
lr1 = SETITEMC(lid1, 'plane', 4);

pc = POPC(lid1, pos);

PUT 'The list item popped from position ' pos ' is: ' pc;

CALL PUTLIST(lid1);

RUN;
```

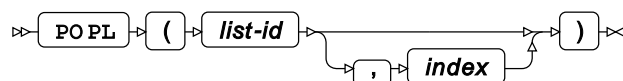
This produces the following output:

```
The list item popped from position 3 is: bicycle
('car' 'train' 'plane' ) [1]
```

The list created by `CALL PUTLIST` shows that the item with the value `bicycle` has been popped, and is no longer in the list.

POPL

Returns and removes a sublist from a specified list.



The function returns the identifier of a sublist at a specified position in a list, and removes that list identifier from the list. This is known as popping, and such an item has been popped from the list. Sublist identifiers are items in the list, and the identifier to be returned is specified by its position in the list.

Return type: List

The list identifier of the sublist.

list-id

Type: List

The identifier for the list.

This is the identifier for the list from which you want to return the sublist identifier.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list from which to pop the item. The default is 1. If you specify a value greater than the number of items in the list, an error is returned.

If you try to pop an item that is not a list, an error message is returned.

Note:

Because popped items are removed from the list, you should be careful when using a loop to set the index for the item to be popped.

Example

In this example, a new list is created from a saved list. The contents of the new list are displayed with `CALL PUTLIST`. The sublist at the third position is then popped. The list identifier for the sublist is written to the log, and `CALL PUTLIST` is again used to display the list.

```
libname temp 'c:\temp';
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  fl = FILLIST('CATALOG', 'temp.listtest.newlist.slist', lid1);

  PUT 'The filled list contains: ';

  CALL PUTLIST(lid1,,0);

  pl = POPL(lid1, 3);

  PUT;
  PUT 'The identifier popped is: ' pl;
  PUT;
  PUT 'The list after popping contains: ';

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
The filled list contains:
(VAR1='ship'
 VAR2='plane'
 (LST1='car'
  LST2='train'
  LST3='bicycle'
 ) [2]
 VAR3='horse'
 VAR4='taxi'
) [1]

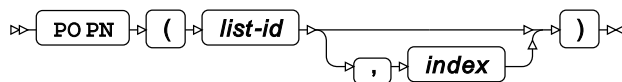
The identifier popped is: 2

The list after popping contains:
(VAR1='ship'
 VAR2='plane'
 VAR3='horse'
 VAR4='taxi'
) [1]
```

The second `PUTLIST` routine shows that the sublist has been removed from the list. The list identifier returned for the sublist is 2. If `index` had not been specified (therefore defaulting to 1), or any other value but 3 had been specified, no list would have been popped, and an error message would have been written to the log.

POPN

Returns and removes a numeric item from a specified list.



The function returns the value of a specified item in a list, and removes that item from the list. This is known as popping, and such an item has been popped from the list. The function returns the next numeric item or a specified numeric item from a list, depending on the options you specify.

Return type: Numeric

list-id

Type: List

The identifier for the list.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list from which to pop the item. The default is 1. If you specify a value greater than the number of items in the list, an error is returned.

If you try to pop an item that is not a number, an error message is returned.

Note:

Because popped items are removed from the list, you should be careful when using a loop to set the index for the item to be popped.

Basic example

In this example, all items are extracted from a list using the `POP`[N](#) (page 1675) function. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMN(lid1, 100, 1);
  rc = SETITEMN(lid1, 200, 2);
  rc = SETITEMN(lid1, 300, 3);
  rc = SETITEMN(lid1, 1001, 4);

  DO y = 1 TO 4;
    pc = POPN(lid1);
    PUT 'List item ' y 'is: ' pc;
  END;

  rc = LISTLEN(lid1);
  PUT 'The list length is now: ' rc;

RUN;
```

This produces the following output:

```
List item 1 is: 100
List item 2 is: 200
List item 3 is: 300
List item 4 is: 1001
The list length is now: 0
```

The list length is now 0, as all items have been popped from the list.

Example – specifying an ordinal position

In this example, an item at a specified position in the list is extracted using the `POPN` [\(page 1675\)](#) function. The result is written to the log.

```
DATA _NULL_;

pos = 3;

lid1 = MAKELIST(4);

lr1 = SETITEMN(lid1, 100, 1);
lr1 = SETITEMN(lid1, 200, 2);
lr1 = SETITEMN(lid1, 300, 3);
lr1 = SETITEMN(lid1, 1001, 4);

pn = POPN(lid1, pos);

PUT 'The list item at position ' pos 'is: ' pn;

CALL PUTLIST(lid1);

RUN;
```

This produces the following output:

```
The list item at position 3 is: 300
(100 200 1001 ) [1]
```

`CALL PUTLIST` is used to show that the item with the value 300 has been removed from the list after it has been popped.

REVLIST

Reverses the order of the items in a list.

➤ **REVLIST** ➤ (➤ *list-id* ➤) ➤

Return type: List

The list identifier.

list-id

Type: List

The identifier for the list.

Example

In this example, the items in a list are reversed. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'train', 2);
  lr1 = SETITEMC(lid1, 'bicycle', 3);
  lr1 = SETITEMC(lid1, 'plane', 4);

  r1 = REVLIST(lid1);

  CALL PUTLIST(lid1);

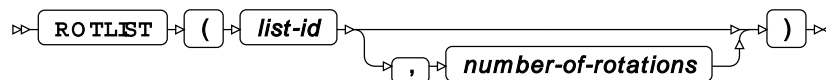
RUN;
```

This produces the following output:

```
('plane' 'bicycle' 'train' 'car' )[1]
```

ROTLIST

Rotates the order of the items in a numeric list by a specified number of characters.



Only lists consisting of numeric items can be rotated.

Return type: List

The list identifier.

list-id

Type: List

The identifier for the list.

number-of-rotations

Optional argument

Type: Numeric

The number of rotations. If not specified, the list is rotated by one position.

A rotation takes the number of characters specified by *number-of-rotations* from the beginning (left-hand) of the list and adds them to the end (right-side) of the list. See the example below.

Example

In this example, the items in a list are rotated by two positions. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 400, 4);

  r1 = ROTLIST(lid1,2);

  call putlist(lid1);

RUN;
```

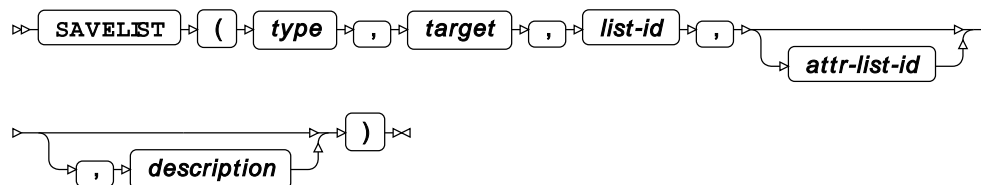
This produces the following output:

```
(300 400 100 200 ) [1]
```

The function has taken the first two entries in the list and moved them to the end of the list.

SAVELIST

Saves a list.



The list can be saved as a catalog entry, or as a file.

Return type: Numeric

type

Type: Character

Specifies the type of file into which the list will be saved. This can be a catalog entry, or a filesystem file.

The type can be of one:

'CATALOG'

The list is written to a catalog entry. Lists can be written to LOG, OUTPUT, SOURCE or SLIST catalog entries. SLIST is a catalog entry provided specifically for lists.

A list that contains sublists can only be saved as an `SLIST` entry.

'FILE'

The list is written to a file.

'FILeref'

The list is written to a file identified by a fileref.

The file or catalog entry into which the list is to be written is specified using *target*.

target

Type: Character

Specifies the filename, fileref or catalog entry name into which you want to write the list. The value should correspond to the type of file you specified in *type*. For example, if the target file is a filesystem file, *type* should be set to `'FILE'`.

list-id

Type: List

The identifier for the list.

attr-list-id

Optional argument

Type: List

The identifier of a list that contains attributes that can be applied to the items in the list specified by *list-id*. For example, you could make an item in a list bold, underlined and blue when it is displayed through certain interfaces or media.

This option has no effect if it is specified for a catalog entry type of `SLIST`.

This option must be specified if the catalog entry type is `LOG`, `OUTPUT` or `SOURCE`.

For information on the attributes that can be set in this list, see below.

description

Optional argument

Type: Character

A description for the list.

The list specified at *attr-list-id* contains codes that define the output format of list items. The code has the format:

```
'type' || 'attributes'x
```

where:

- *type* specifies whether the *attributes* applies to items that define data, headings or titles in the list

- *attributes* specifies attributes for the list items, such as text colour, text weight, underlining, and so on
- *x* defines the entry as a hexadecimal number

For example, if your list had the data `horse` which you wanted to display in blue text with the first character in red, blinking and underlined, you would specify the code `'D' | | '2A10101010'x`.

The list specified by *attr-list-id* must contain the same number of items as are in the list specified by *list-id*.

The following *attributes* are available:

Colour	Code	Attribute	Code
Blue	10	None	00
Red	20	Bold	01
Pink	30	Underline	02
Green	40	Blink	04
Cyan	50	Reverse	08
Yellow	60		
White	70		
Orange	80		
Black	90		
Magenta	A0		
Gray	B0		
Brown	C0		

To create a code for a list item, one or more attribute codes are added to a colour code. For example, to specify that a list item should be displayed in blue and underlined, you would specify for that item the value `12`, that is `10` (blue) plus `02` (underline). To specify that an item is displayed in brown, bold, blinking text specify `C5`. Numbers are in hexadecimal format, so green (`40`), blink (`04`) and reverse (`08`) would be specified as `4C`.

If the list item is longer than one character, and you want the entire string to have one or more attributes, you must set the attributes for each character.

You must also specify whether the list item to which you are applying attributes is a heading, title or data. To do this, you apply a type code before the attribute specification. This code can be:

H	Heading
T	Title
D	Data

Basic example

In this example, a list is created that contains three items. This list is then saved to a folder on the device using a pathname.

```
DATA _NULL_;

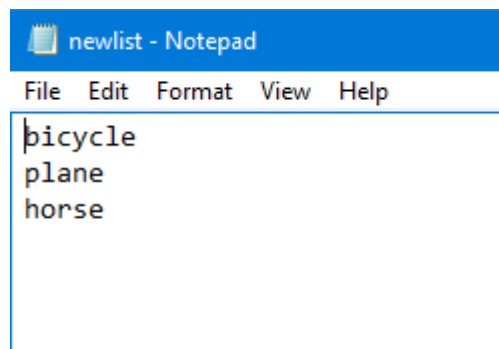
  lid1 = envlist('g');

  lr1 = SETITEMC(lid1, 'bicycle', 1, 'y');
  lr1 = SETITEMC(lid1, 'plane', 2, 'y');
  lr1 = SETITEMC(lid1, 'horse', 3, 'y');

  rx = savelist('FILE', 'c:\temp\newlist', lid1);

RUN;
```

This creates the file `newlist` in the folder `c:\temp`. The file contains the items specified by the `SETITEMC` functions. For example, if the file is opened in Notepad:



Example – saving the list to a fileref

In this example, a list is created that contains three items. This list is then saved to a file specified using a fileref.

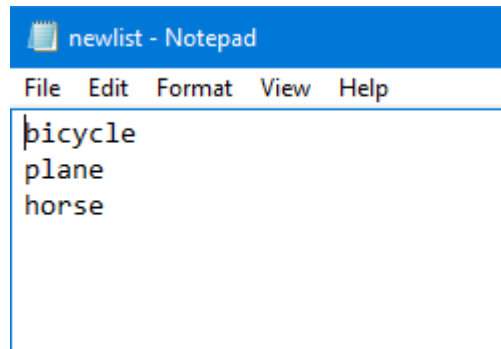
```
filename listf 'C:\temp\newlist';
DATA _NULL_;
  lid1 = MAKELIST(3, 'g');

  lr1 = SETITEMC(lid1, 'bicycle', 1, 'y');
  lr1 = SETITEMC(lid1, 'plane', 2, 'y');
  lr1 = SETITEMC(lid1, 'horse', 3, 'y');

  rx = savelist('FILEREf', 'listf', lid1);

RUN;
```

This creates the file `newlist` in the folder `c:\temp`. The file contains the items specified by the `SETITEMC` functions. For example, if the file is opened in Notepad:



Example – saving the list to a catalog entry and creating an attribute list

In this example, a list is created that contains three items. This list is then saved to a catalog entry.

```
libname temp 'C:\temp';
DATA _NULL_;

    lid1 = MAKELIST(3, 'g');

    lr1 = SETITEMC(lid1, 'bicycle', 1, 'Y');
    lr1 = SETITEMC(lid1, 'plane', 2, 'Y');
    lr1 = SETITEMC(lid1, 'horse', 3, 'Y');

    lid2 = envlist('g');

    lr2 = SETITEMC(lid2, 'D' || '10101010101010'x, 1, 'Y');
    lr2 = SETITEMC(lid2, 'D' || '2110101010'x, 2, 'Y');
    lr2 = SETITEMC(lid2, 'D' || '5C10101010'x, 3, 'Y');

    rx = savelist('CATALOG', 'temp.listtest.newlist.source', lid1, lid2);

RUN;
```

This creates the catalog entry of the type `source` in the catalog `listtest`. The file contains the items specified by the `SETITEMC` functions, with the formatting specified by the list items in the second list.

Example – appending the list to a fileref

In this example, a list is created that contains three items. The list is then saved to a specified file that already exists.

```
filename listf 'C:\temp\newlist';
DATA _NULL_;

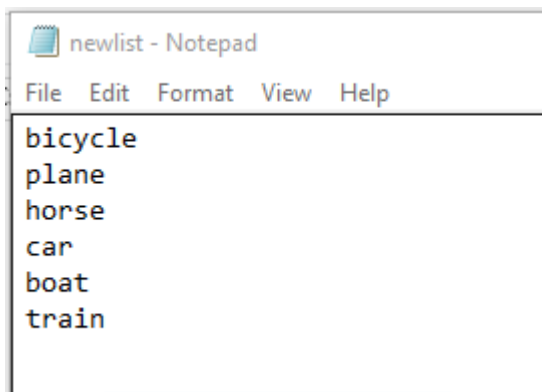
  lid1 = MAKELIST(3, 'g');

  lr1 = SETITEMC(lid1, 'car', 1, 'y');
  lr1 = SETITEMC(lid1, 'boat', 2, 'y');
  lr1 = SETITEMC(lid1, 'train', 3, 'y');

  sl = savelist('FILEREFS<APPEND>', 'listf', lid1);

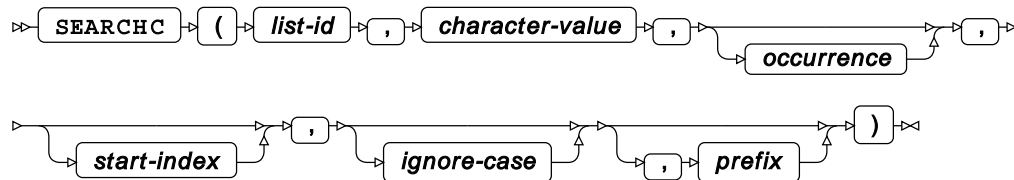
RUN;
```

This appends the list items specified by the SETITEMC functions to the file `newlist` in the folder `c:\temp`. For example, if the file is opened in Notepad:



SEARCHC

Returns the ordinal position of a string in a list that matches the specified character or string.



The search and target strings must match completely, unless you specify a search only for a prefix. You can set a start position in the list from which to search. If the list contains more than one occurrence of the string, you can specify an occurrence from which to search.

Return type: Numeric

The ordinal position of the item in the list, if found; otherwise 0 (zero).

list-id

Type: List

The identifier for the list.

character-value

Type: Character

The character or string to find in the list.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching. The default is 1.

ignore-case

Optional argument

Specifies whether case should be ignored in the search.

"N"

Case is not ignored.

"Y"

Case is ignored. This is the default.

prefix

Optional argument

Specifies whether *character-value* is a prefix or the entire character or string to be found.

"N"

character-value defines the character or string to be found. This is the default.

"Y"

character-value defines a prefix to be found.

By default, this function searches for strings that exactly match *character-value*. For example, if your list contains the values `train_carriage` and `train_locomotive`, and you search for `train`, neither value will be found. If you set this option to `Y` and specify `train_` in *character-value*, then both values will match.

Basic example

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. `SEARCHC` is then used to find the position of the string `train`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMC(lid1, 'car', 1);
  rc = SETITEMC(lid1, 'bicycle', 2);
  rc = SETITEMC(lid1, 'train', 3);
  rc = SETITEMC(lid1, 'plane', 4);

  ni = SEARCHC(lid1,'train');
  PUT 'The item is at position ' ni;

RUN;
```

This produces the following output:

```
The item is at position 3
```

Example – searching for an occurrence from a specified point

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. `SEARCHC` is then used to find the position of the second occurrence of the string `train`, starting from position three and ignoring case. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(6);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'train', 2);
  lr1 = SETITEMC(lid1, 'bicycle', 3);
  lr1 = SETITEMC(lid1, 'train', 4);
  lr1 = SETITEMC(lid1, 'plane', 5);
  lr1 = SETITEMC(lid1, 'train', 6);

  ni = SEARCHC(lid1,'train',2, 3);
  PUT 'The item is at position ' ni;

RUN;
```

This produces the following output:

```
The item is at position 6
```


The second occurrence of train in this search is the sixth item in the list. The occurrence of the string at the second position in the list is ignored, as the search started at the third index position.

Example – searching for an occurrence in a specified case

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. `SEARCHC` is then used to find the position of the occurrence of the string `TRAIN`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(6);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'train', 2);
  lr1 = SETITEMC(lid1, 'bicycle', 3);
  lr1 = SETITEMC(lid1, 'train', 4);
  lr1 = SETITEMC(lid1, 'plane', 5);
  lr1 = SETITEMC(lid1, 'TRAIN', 6);

  ni = SEARCHC(lid1,'TRAIN',,, 'N');
  PUT 'The item is at position ' ni;

RUN;
```

This produces the following output:

```
The item is at position 6
```

The case of the string is ignored; therefore, `TRAIN` is the same as `train`, and the second occurrence in this search is the sixth item in the list. The occurrence of the string at the second position in the list is ignored, as the search started at index position three.

Example – searching for a prefix

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. `SEARCHC` is then used to find the position of each occurrence of an item that starts with the substring `train`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMC(lid1, 'train_locomotive', 1);
  lr1 = SETITEMC(lid1, 'train_carriage', 2);
  lr1 = SETITEMC(lid1, 'car', 3);
  lr1 = SETITEMC(lid1, 'plane', 4);

  DO i = 1 TO listlen(lid1);
    ni = 0;
    ni = SEARCHC(lid1,'train_',,i,, 'y');
    IF ni > 0 THEN PUT 'An item is at position ' ni;
  END;

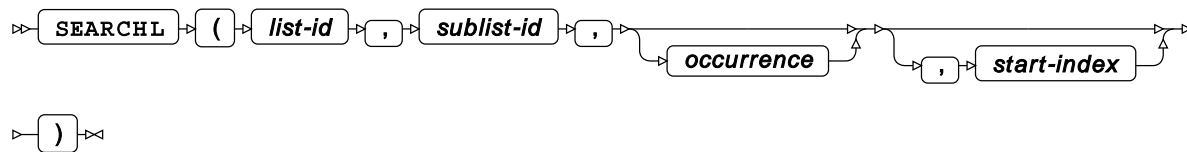
RUN;
```

This produces the following output:

```
An item is at position 1
An item is at position 2
```

SEARCHL

Returns the position of a specified sublist in a list.



By default, this function searches for the first occurrence of the specified sublist, starting from the first position in the list. You can, however, search for a specified occurrence, or start at a specified position.

Return type: Numeric

list-id

Type: List

The identifier for the list.

sublist-id

Type: List

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the list to find, if it exists. This must be greater than 0 (zero). This must be greater than 0 (zero). For example, to find the second occurrence of the list, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching. The default is 1.

Basic example

In this example, a list is created using the [MAKENLIST](#) (page 1667) and [SETNITEMC](#) (page 1696) functions. A second list is similarly created, and then inserted in the first list using [INSERTL](#) (page 1656). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  rc = SETNITEMC(lid1, 'ship', 'var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  rc = SETNITEMC(lid2, 'car', 'lst1');
  rc = SETNITEMC(lid2, 'train', 'lst2');
  rc = SETNITEMC(lid2, 'bicycle', 'lst3');

  rc=INSERTL(lid1,lid2,2);

  rc = SEARCHL(lid1, lid2);

  PUT 'The sublist is at position: ' rc;

RUN;
```

This produces the following output:

```
The sublist is at position: 2
```

Example – finding an occurrence of a sublist, and finding after a position

In this example, a list is created using the [MAKENLIST](#) (page 1667) and [SETNITEMC](#) (page 1696) functions. A second list is similarly created, and then inserted in the first list using [INSERTL](#) (page 1656). The list is inserted twice, first at the second position in the list, and then again at the fourth position. The sublist is then found by starting from a specified position, and then by looking for a specified occurrence. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  v = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  lr2 = SETNITEMC(lid2, 'car', 'lst1');
  lr2 = SETNITEMC(lid2, 'train', 'lst2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'lst3');
```

```

il1 = INSERTL(lid1,lid2,2);
il2 = INSERTL(lid1,lid2,4);

slr = SEARCHL(lid1, lid2,2);
PUT 'The sublist is at position: ' slr;

slr2 = SEARCHL(lid1, lid2,,3);
PUT 'The sublist is at position: ' slr2;

RUN;

```

This produces the following output:

```

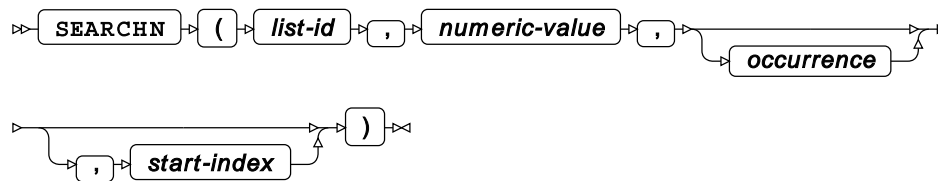
The sublist is at position: 4
The sublist is at position: 4

```

In this example, 4 is returned for both search methods. The second occurrence of the list is at position four; and the first occurrence of the list after index position two is also at position four.

SEARCHN

Returns the ordinal position of a specified number.



The search and target numbers must match completely. You can set a start position in the list from which to search. If the list contains more than one occurrence of the number, you can specify an occurrence from which to search.

Return type: Numeric

The ordinal position of the item in the list, if found; otherwise 0 (zero).

list-id

Type: List

The identifier for the list.

numeric-value

Type: Numeric

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching. The default is 1.

Basic example

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMN](#) (page 1703) functions. `SEARCHN` is then used to find the position of the string 200. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(6);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 400, 4);
  lr1 = SETITEMN(lid1, 500, 5);
  lr1 = SETITEMN(lid1, 600, 6);

  ni = SEARCHN(lid1,400);
  PUT 'The item is at position ' ni;

RUN;
```

This produces the following output:

```
The item is at position 4
```

Example – searching for an occurrence from a specified point

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMN](#) (page 1703) functions. [SEARCHN](#) is then used to find the position of the third occurrence of the string 400, starting from and including the second position in the list. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(6);

  lr1 = SETITEMN(lid1, 400, 1);
  lr1 = SETITEMN(lid1, 400, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 400, 4);
  lr1 = SETITEMN(lid1, 500, 5);
  lr1 = SETITEMN(lid1, 400, 6);

  ni = SEARCHN(lid1,400,2,3);
  PUT 'The item is at position ' ni;

RUN;
```

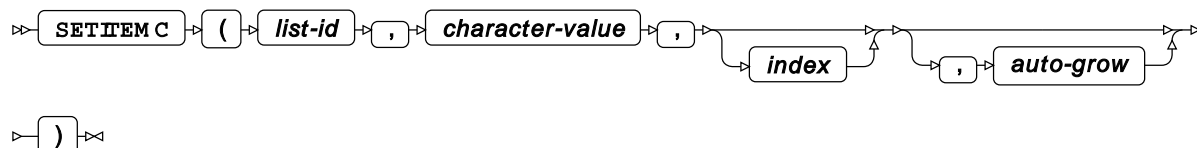
This produces the following output:

```
The item is at position 6
```

There are four occurrences of the specified number in this list; the third occurrence, starting from the second position in the list, is at position six.

SETITEMC

Sets the specified list item to the specified character string.



The character or string specified overwrites the item at the specified position. If no position is specified, the character or string overwrites the item at the first position in the list.

If you want to insert a character or string without overwriting any items, see [INSERTC](#) (page 1654).

Return type: Numeric

Returns the list identifier.

list-id

Type: List

The identifier for the list.

character-value

Type: Character

The value to which to set this list item. This must be a character or string.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to place *character-value*.

If you do not specify a value for this argument, the first item in the list is set to *character-value*.

If you specify a value for this argument that is greater than the number of items specified when the list was created, an error message is returned, unless *auto-grow* is set to \mathbb{Y} .

auto-grow

Optional argument

Specifies that the number of items in the list can increase beyond that specified when the list was created.

"N"

The list cannot grow beyond the number of items specified when it was created. This is the default.

"Y"

The list can grow beyond the number of items specified when it was created.

You can set *index* to any number larger than the length of *list-id*, as long as \mathbb{Y} is also specified for *auto-grow*. If *index* is greater than the length of *list-id* + 1, then the list items between the original last index item and the new last index item contain text missing values. For an example, see *Example using auto-grow* below.

Basic example

In this example, a new list is created that can hold four items. The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMC(lid1, 'a', 1);
  rc = SETITEMC(lid1, 'b', 2);
  rc = SETITEMC(lid1, 'c', 3);
  rc = SETITEMC(lid1, 'd', 4);

  DO y = 1 TO 4;
    pc = GETITEMC(lid1, y);
    PUT 'List item ' y 'is: ' pc;
  END;

RUN;
```

This produces the following output:

```
List item 1 is: a
List item 2 is: b
List item 3 is: c
List item 4 is: d
```

Example using *auto-grow*

In this example, a new list is created that can hold four items, but is extended to six items by using the *auto-grow* argument with `SETITEMC` [\(page 1696\)](#). The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). The `LISTLEN` [\(page 1664\)](#) function is also used to get the length of the new list. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMC(lid1, 'a', 1);
  lr1 = SETITEMC(lid1, 'b', 2);
  lr1 = SETITEMC(lid1, 'c', 3);
  lr1 = SETITEMC(lid1, 'd', 4);

  rc = SETITEMC(lid1, 'g', 7, 'Y');

  DO y = 1 TO LISTLEN(lid1);
    pc = GETITEMC(lid1, y);
    PUT 'List item ' y 'is: ' pc;
  END;

RUN;
```

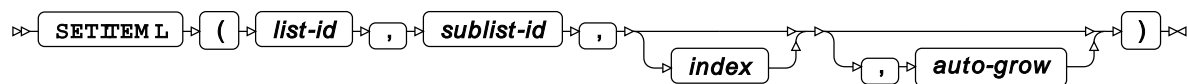

This produces the following output:

```
List item 1 is: a
List item 2 is: b
List item 3 is: c
List item 4 is: d
List item 5 is:
List item 6 is:
List item 7 is: g
```

The list has been extended to consist of seven items, with items five and six set as missing.

SETITEML

Sets the specified list item to the specified sublist.



The sublist specified overwrites the item at the specified position. If no position is specified, the sublist overwrites the item at the first position in the list. It is not the sublist itself that is placed in the list, but a pointer to it, identified in other functions by the list identifier.

If you want to insert a list without overwriting any items, see [INSERTL](#) (page 1656).

Return type: Numeric

The identifier of the list into which the sublist is written.

list-id

Type: List

The identifier for the list.

sublist-id

Type: List

The identifier of the sublist that overwrites the item.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to place *sublist-id*.

If you do not specify a value for this argument, the first item in the list is set to *sublist-id*

If *index* is greater than the number of items in the list an error message is returned, unless you specify *auto-grow*.

auto-grow

Optional argument

Specifies that the number of items in the list can increase beyond that specified when the list was created.

"N"

The list cannot grow beyond the number of items specified when it was created. This is the default.

"Y"

The list can grow beyond the number of items specified when it was created.

```
lid1 = MAKELIST(9);
```

The list can therefore contain nine items. If *auto-grow* is set to Y and index is set to 6:

```
rc = SETITEML(lid1, lid2, 6, 'y');
```

then the list replaces whatever is currently at the specified index position. However, if *auto-grow* is set to Y and index is set to 12:

```
rc = SETITEMC(lid1, lid2, 12, 'y');
```

then the list grows to contain twelve items. The list occupies the twelfth position in the list, and positions ten and eleven are set to missing values.

You can set *index* to any number larger than the length of *list-id*, as long as Y is also specified for *auto-grow*. If *index* is greater than the length of *list-id* + 1, then the list items between the original last index item and the new last index item contain empty lists with list identifiers set to 0 (zero). For an example, see *Example – inserting sublist after last position*.

Basic example

In this example, a new list is created that can hold four items. A second list is similarly created. This function is then used to set the first item in the list to the sublist. Because *index* is not specified, the sublist overwrites the first item in the list. The first list is written to the log using `CALL PUTLIST` [↗](#) (page 1737), and the other results generated by the example are also written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  lr2 = SETNITEMC(lid2, 'car', 'lst1');
  lr2 = SETNITEMC(lid2, 'train', 'lst2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'lst3');
```

```

sir = SETITEML(lid1,lid2);
PUT 'The identifier of the list: ' sir;

gir = GETITEML(lid1,1);
PUT 'The identifier of the sublist: ' gir;

CALL PUTLIST(lid1);

RUN;

```

This produces the following output:

```

The identifier of the list: 1
The identifier of the sublist: 2
(VAR1=(LST1='car' LST2='train' LST3='bicycle' ) [2] VAR2='plane' VAR3='horse'
VAR4='taxi' ) [1]

```

In the list written by `CALL PUTLIST` you can see that the sublist has replaced the first item in the list.

Example – putting sublist at a specified position

In this example, a new list is created that can hold four items. A second list is similarly created. The specified sublist is then inserted after the last item in the list. The first list is written to the log using `CALL PUTLIST` [\(page 1737\)](#), and the other results generated by the example are also written to the log.

```

DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'vv1', 'vv2', 'vv3');

  lr2 = SETNITEMC(lid2, 'car', 'vv1');
  lr2 = SETNITEMC(lid2, 'train', 'vv2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'vv3');

  sir = SETITEML(lid1,lid2,3);

  CALL PUTLIST(lid1);

RUN;

```

This produces the following output:

```

(VAR1='ship'
VAR2='plane'
VAR3=(VV1='car'
      VV2='train'
      VV3='bicycle'
    ) [2]
VAR4='taxi'
) [1]

```

In the list written by `CALL PUTLIST` you can see that the sublist has replaced the third item in the list.

Example – inserting sublist after last position

In this example, a new list is created that can hold four items. A second list is similarly created. The specified sublist then overwrites the item at the specified position. The first list is written to the log using `CALL PUTLIST` [↗](#) (page 1737), and the other results generated by the example are also written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  lr2 = SETNITEMC(lid2, 'car' , 'lst1');
  lr2 = SETNITEMC(lid2, 'train', 'lst2');
  lr2 = SETNITEMC(lid2, 'bicycle', 'lst3');

  sir = SETITEML(lid1, lid2, 5, 'y');

  CALL PUTLIST(lid1, , 0);

RUN;
```

This produces the following output:

```
(VAR1='ship'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
(LST1='car'
LST2='train'
LST3='bicycle'
) [2]
) [1]
```

In the list written by `CALL PUTLIST` you can see that the sublist has become the fifth item in the list.

If, however, you set the insertion point to index position seven:

```
rc = SETITEML(lid1, lid2, 7, 'y');
```

then the following output would be produced:

```
(VAR1='ship'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
<invalid list id>[0]
<invalid list id>[0]
(LST1='car'
LST2='train'
LST3='bicycle'
)[2]
)[1]
```

The sublist has become the seventh item in the list, and two intervening invalid sublists have been created, with the identifier 0. No error message is generated by WPS.

You can get the identifier for the invalid list. For example if you specified `GETITEML` as follows:

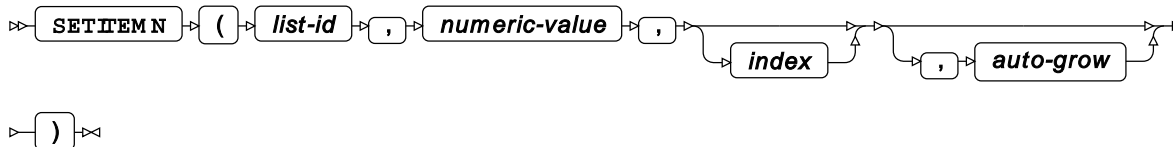
```
rc = GETITEML(lid1,6);
PUT 'The identifier for the sublist: ' rc;
```

and inserted it after the `CALL PUTLIST` in the example above, the following output would be produced:

```
(VAR1='ship'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
<invalid list id>[0]
<invalid list id>[0]
(LST1='car'
LST2='train'
LST3='bicycle'
)[3]
)[2]
The identifier for the sublist: 0
```

SETITEMN

Sets the specified list item to the specified number.



The number specified overwrites the item at the specified position. If no position is specified, the number overwrites the item at the first position in the list.

If you want to insert a number without overwriting any items, see [INSERTN](#) (page 1659).

Return type: Numeric

Returns the list identifier.

list-id

Type: List

The identifier for the list.

numeric-value

Type: Numeric

The value to which to set this list item. This must be a number.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to place *numeric-value*.

If you do not specify a value for this argument, the first item in the list is set to *numeric-value*.

If you specify a value for this argument that is greater than the number of items specified when the list was created, an error message is returned, unless *auto-grow* is set to *Y*.

auto-grow

Optional argument

Specifies that the number of items in the list can increase beyond that specified when the list was created.

"N"

The list cannot grow beyond the number of items specified when it was created. This is the default.

"Y"

The list can grow beyond the number of items specified when it was created.

You can set *index* to any number larger than the length of *list-id*, as long as *Y* is also specified for *auto-grow*. If *index* is greater than the length of *list-id* + 1, then the list items between the original last index item and the new last index item contain numeric missing values. For an example, see *Example using auto-grow* below.

Basic example

In this example, a new list is created that can hold four items. The items are then extracted from the list using `GETITEMN` (page 1631). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 1001, 4);

  DO y = 1 TO 4;
    pc = GETITEMN(lid1, y);
    PUT 'List item ' y 'is: ' pc;
  END;

RUN;
```

This produces the following output:

```
List item 1 is: 100
List item 2 is: 200
List item 3 is: 300
List item 4 is: 1001
```

Example – using *auto-grow*

In this example, a new list is created that can hold four items, but is extended to six items by using the *auto-grow* argument with `SETITEMN` (page 1703). The items are then extracted from the list using `GETITEMN` (page 1631). The `LISTLEN` (page 1664) function is also used to get the length of the new list. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 200, 2);
  lr1 = SETITEMN(lid1, 300, 3);
  lr1 = SETITEMN(lid1, 1001, 4);

  sir = SETITEMN(lid1, 50, 7, 'Y');

  DO y = 1 TO LISTLEN(lid1);
    git = GETITEMN(lid1, y);
    PUT 'List item ' y 'is: ' git;
  END;

RUN;
```

This produces the following output:

```
List item 1 is: 100
List item 2 is: 200
List item 3 is: 300
List item 4 is: 1001
List item 5 is: .
List item 6 is: .
List item 7 is: 50
```

The list has been extended to consist of seven items, with items five and six missing.

SETLATTR

Set the attributes for a list or for specified items in a list.



Some attributes can only be set for lists, not for list items.

Return type: Numeric

list-id

Type: List

The identifier for the list.

attribute-list

A list of one or more attributes to apply to the specified list or to a specified item in that list. Attributes are separated by spaces. The list must be enclosed by quotation marks.

"ACTIVE"

Currently unsupported. Applies to list items only.

"ANYNAMES"

A list item name can have any format. This is the default. For example, the name can be longer than eight characters, can start with a numeric, can contain non-alphanumeric characters, and so on. To restrict item names to SAS format only, see [SASNAMES](#).

Applies to lists only.

"AUTO"

Currently unsupported. Applies to list items only.

"CHARONLY"

The list can only be assigned character values. If you assign a numeric value, an error occurs and a message is written to the log. See also `NOCHARONLY`.

Applies to lists only.

"COPY"

Sublists are included in lists as list items, rather than as list identifiers. This is the default.

Applies to lists only.

"DELETE"

Items can be deleted. This is the default. See also `NODELETE`.

Applies to lists and to list items.

"DUPNAMES"

Duplicate item names are allowed in lists. This is the default. If you want to ensure duplicate item names cannot be set for a list, see `NODUPNAMES`.

Applies to lists only.

"FIXEDLENGTH"

The number of items in a list is fixed when it is created. Items cannot be added to or deleted from the list. If you want to add or delete items, see `NOFIXEDLENGTH`, which is the default setting.

Applies to lists only.

"FIXEDTYPE"

Specifies that the type (character, numeric or list) of the items in a list is set when using a list function, and items of other types cannot be written to list items. For example, if you specify a numeric value for a list item that was originally specified as a character:

```
rc = SETITEMN(lid1, 100, 1);
```

then, by default WPS, automatically converts it to a character value, and a note is written to the log indicating that this has occurred. If you have previously set this attribute for the list or list item, the value will instead be set to a missing value, and an error message and note will be written to the log.

You reset this attribute using see `NOFIXEDTYPE` (this is the default).

Applies to lists and to list items.

"HONORCASE"

The case of list item names is honoured. For example, `VAR1` is different to `var1` and to `Var1`. By default case is not honoured (see `NOHONORCASE`).

Applies to lists only.

"NOHONORCASE"

The case of item names is not honoured. For example, the names `VAR1`, `var1` and `Var1` are considered the same. This is the default. See also `HONORCASE`.

Applies to lists only.

"INACTIVE"

Currently unsupported.

Applies to lists items only.

"NOAUTO"

Currently unsupported. Applies to list items only.

"NOCHARONLY"

Lists can contain any type of value. This is the default. To specify that lists can only contain character values, specify `CHARONLY`

Applies to lists only.

"NOCOPY"

Sublists are output as list identifiers. This is the default. To enable functions to output sublists as entries in the containing list, specify `COPY`

Applies to lists only.

"NODELETE"

List items cannot be deleted. See also `DELETE`.

Applies to lists and to list items.

"NODUPNAMES"

Duplicate list item names are not allowed. For example:

```
DATA _NULL_;

  lid1 = MAKENLIST('1', 'VAR4', 'VAR2', 'VAR3', 'VAR4');

  rc = SETLATTR(lid1, 'nodupnames');

  rc = SETNITEMC(lid1, 'bicycle', 'var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'plane', 'var3');
  rc = SETNITEMC(lid1, 'horse', 'var4');
  rc = SETNITEMC(lid1, 'taxi', 'var4', 2);

  CALL PUTLIST(lid1, 0);

RUN;
```

causes a message to be written to the log warning that the item with the duplicate name cannot be set. The list only contains the first four items.

Applies to lists only.

If the list is defined with duplicate item names, then attempting to set this attribute causes an error message. For example, if the list in the above example had been specified as:

```
lid1 = MAKENLIST('l', 'VAR1', 'VAR2', 'VAR3', 'VAR4', 'VAR4');
```

then running the **DATA** step would cause a message to be written to the log informing you that the attribute could not be set. The list is then created as if **DUPNAMES** is specified. See also **DUPNAMES**.

"NOFIXEDLENGTH"

The length of the list is not fixed. Items can be added to or deleted from the list. This attribute applies to lists. This is the default. See also **FIXEDLENGTH**.

Applies to lists only.

"NOFIXEDTYPE"

Specifies that the type (character, numeric or list) is not fixed, and functions that write to the items do not have to be of the same type. For example, if you specify a numeric value to a character list item:

```
rc = SETITEMC(lid1, 100, 1);
```

then, by default WPS, automatically converts it to a character value. A note would be written to the log indicating that the numeric value has been converted to a character value.

This is the default value. If you want to ensure list items can only have particular types, specify **FIXEDTYPE**.

"NONUMONLY"

Specified for lists only. Disables **NUMONLY**, if it has been set. This is the default.

Applies to lists only.

"NOUPDATE"

The values of items cannot be changed or updated. The positioning of this option affects the results. For example, in this code fragment:

```
lid1 = MAKENLIST('l', 'lvar', 'var2', 'var3', 'var4');

rc = SETLATTR(lid1, 'noupdate');

rc = SETNITEMN(lid1, 1, 'lvar');
rc = SETNITEMC(lid1, 'plane', 'var2');
rc = SETNITEMN(lid1, 200, 'var3');
rc = SETNITEMN(lid1, 300, 'VAR4');
```

no values are set, and a message is returned to the log, because the attribute prevents updates to the list. In this code fragment:

```
lid1 = MAKENLIST('l', 'lvar', 'var2', 'var3', 'var4');

rc = SETNITEMN(lid1, 1, 'lvar');
rc = SETNITEMC(lid1, 'plane', 'var2');
rc = SETNITEMN(lid1, 200, 'var3');
rc = SETNITEMN(lid1, 300, 'VAR4');

rc = SETLATTR(lid1, 'nouupdate');

rc = SETNITEMN(lid1, 2, 'lvar');
```

The list items are set to the values specified. However, the final `SETNITEMN` cannot change the value of the specified item, as the attribute was set before it. An error message is returned, and the value of the item is not updated. See also `UPDATE`.

Applies to lists and to list items.

"NOWRITE"

The list item cannot be saved to a file or catalog using `SAVELIST` [↗](#) (page 1683). To enable a list item to be saved, specify `WRITE` (this is the default).

Applies to list items only. If you do not specify an index position, this attribute is ignored.

"NUMONLY"

Items in the list can only be numeric. If you specify a character value to a list item, an error message is returned.

To switch off this attribute, specify `NONUMONLY` (the default).

Applies to lists only.

"SASNAMES"

A list item name must be in SAS naming format. That is, the name must contain eight characters or fewer, can only contain alphanumeric characters, and cannot start with a numeric. For example, if this attribute is set the list items names `lvar`, `var^` and `varvar123` would be invalid.

If you specify a name that does not conform to SAS naming format, then `SETLATTR` cannot subsequently be set and an error message is returned.

index cannot be specified for this attribute; if it is, the attribute is not set, and defaults to `ANYNAMES`.

Applies to lists only.

"UPDATE"

Default for lists and list items. List items can be updated. See also `NOUPDATE`.

Applies to lists and to list items.

"WRITE"

Enables a list item to be saved in a file or catalog using `SAVELIST` [↗](#) (page 1683). This is the default. To switch off this attribute, specify `NOWRITE`.

Applies to list items only. If you do not specify an index position, this attribute is ignored.

index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list of an item.

The following attributes are set by default for a list:

DELETE	UPDATE	NOFIXEDTYPE	NOFIXEDLENGTH	ANYNAMES
DUPNAMES	NOCHARONLY	NONUMONLY	COPY	NOHONORCASE

The following attributes are set by default for a list item:

ACTIVE	WRITE	NOAUTO	DELETE	UPDATE	NOFIXEDTYPE
--------	-------	--------	--------	--------	-------------

Example

In this example, a list is created using the `MAKELIST` [↗](#) (page 1667) and `SETITEMC` [↗](#) (page 1696) functions. Attributes are then set for the specified list, and for an item in the list. These attributes are returned using `GETLATTR`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'ship', 'var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  attr4list = GETLATTR(lid1);
  PUT 'All default attributes for list: ' attr4list;

  pq = SETLATTR(lid1,'FIXEDLENGTH CHARONLY');

  attr4list = GETLATTR(lid1);
  PUT 'Modified attributes for list: ' attr4list;

  attr4item = GETLATTR(lid1,2);
  PUT 'Default attributes for second item: ' attr4item;

  pq = SETLATTR(lid1,'NOWRITE AUTO',2);

  attr4item = GETLATTR(lid1,2);
  PUT 'Modified attributes for second item: ' attr4item;

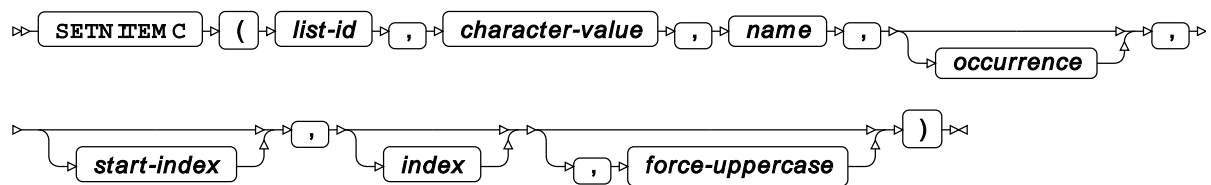
RUN;
```

This produces the following output:

```
All default attributes for list:
DELETE UPDATE NOFIXEDTYPE NOFIXEDLENGTH ANYNAMES DUPNAMES NOCHARONLY NONUMONLY COPY
NOHONORCASE
Modified attributes for list:
DELETE UPDATE NOFIXEDTYPE FIXEDLENGTH ANYNAMES DUPNAMES CHARONLY NONUMONLY COPY
NOHONORCASE
Default attributes for second item: ACTIVE WRITE NOAUTO DELETE UPDATE NOFIXEDTYPE
Modified attributes for second item: ACTIVE NOWRITE AUTO DELETE UPDATE NOFIXEDTYPE
```

SETNITEMC

Sets the specified named list item to the specified character string.



The character or string specified overwrites the specified named item.

If you want to insert a character or string without overwriting any items, see [INSERTC](#) (page 1654), which can be used to insert a character or string at a specified position in a list.

Return type: Numeric

The list identifier.

list-id

Type: List

The identifier for the list.

character-value

Type: Character

The value to which to set this list item. This must be a character or string.

name

Type: Character

The name of the list item for which you want to set a value.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item to set. This must be greater than 0 (zero). The default is 1. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

index

Optional argument

Type: Var

The name of a variable in which the position of the item specified is returned. If the value of this argument is anything except a variable name, an error is returned.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, a new list is created that contains four named items. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
) [1]
```

The output shows that the values have been assigned to the specified items. The item names are displayed in upper case in the output.

In this example, the case in which you specify the item names is ignored. You could, for example, have specified:

```
rc = SETNITEMC(lid1, 'bicycle','VAR1');
rc = SETNITEMC(lid1, 'plane', 'var2');
rc = SETNITEMC(lid1, 'horse', 'Var3');
rc = SETNITEMC(lid1, 'taxi', 'var4');
```

The output would still be the same.

Example – including unspecified item names in the list

In this example, a new list is created that can hold four named items, but is extended to five items using `SETNITEMC` [\(page 1712\)](#). `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  rc = SETNITEMC(lid1, 'bicycle','var1');
  rc = SETNITEMC(lid1, 'plane', 'var2');
  rc = SETNITEMC(lid1, 'horse', 'var3');
  rc = SETNITEMC(lid1, 'taxi', 'var4');
  rc = SETNITEMC(lid1, 'ship', 'var5');

  CALL PUTLIST(lid1,,0);

RUN;
```


This produces the following output:

```
(VAR1='bicycle'  
VAR2='plane'  
VAR3='horse'  
VAR4='taxi'  
var5='ship'  
) [1]
```

The output shows that values have been assigned to the specified items, and a new item, named `var5`, has been added to the list. On display, the case of the new item name matches the case with which it was specified.

Example – non-unique item names

In this example, a new list is created that can hold five named items, where two of the items have non-unique item names. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4');  
  
  rc = SETNITEMC(lid1, 'bicycle','var1');  
  rc = SETNITEMC(lid1, 'plane', 'var2');  
  rc = SETNITEMC(lid1, 'horse', 'var3');  
  rc = SETNITEMC(lid1, 'taxi', 'var4');  
  rc = SETNITEMC(lid1, 'ship', 'var4',2);  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1='bicycle'  
VAR2='plane'  
VAR3='horse'  
VAR4='taxi'  
VAR4='ship'  
) [1]
```

The output shows that values have been assigned to the specified items, and that the items with the same name have unique values because the occurrence for the item has been specified. If the occurrence is not specified:

```
rc = SETNITEMC(lid1, 'bicycle','var1');  
rc = SETNITEMC(lid1, 'plane', 'var2');  
rc = SETNITEMC(lid1, 'horse', 'var3');  
rc = SETNITEMC(lid1, 'taxi', 'var4');  
rc = SETNITEMC(lid1, 'ship', 'var4');
```

the following output is produced:

```
(VAR1='bicycle'  
VAR2='plane'  
VAR3='horse'  
VAR4='ship'  
VAR4=.  
) [1]
```

The first occurrence of the item with the specified name is overwritten, and the second occurrence of the item has a missing value.

In this example, three items with non-unique names are specified:

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4','var4');  
  
  lr1 = SETNITEMC(lid1, 'bicycle','var1');  
  lr1 = SETNITEMC(lid1, 'plane', 'var2');  
  lr1 = SETNITEMC(lid1, 'horse', 'var3');  
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');  
  lr1 = SETNITEMC(lid1, 'ship', 'var4');  
  lr1 = SETNITEMC(lid1, 'car', 'var4',2);  
  lr1 = SETNITEMC(lid1, 'roller skates', 'var4',3);  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1='bicycle'  
VAR2='plane'  
VAR3='horse'  
VAR4='ship'  
VAR4='car'  
VAR4='roller skates'  
) [1]
```

The first occurrence of the item with the specified name is overwritten, the second and third occurrences of the item have the occurrence specified, and are added to the list in the appropriate positions..

Example – starting from an index position

In this example, a new list is created that can hold six named items, where three of the items have non-unique item names. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4', 'var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');
  lr1 = SETNITEMC(lid1, 'car', 'var4',2);
  lr1 = SETNITEMC(lid1, 'ship', 'var4',2,5);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
VAR4='car'
VAR4='ship'
)[1]
```

The output shows that values have been assigned to the specified items. The third occurrence of the item named `var4` has the correct value assigned because the function searched for the item name starting from the fifth index position. The second occurrence from that starting position has the value `ship`.

Example – specifying case

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that holds four named items. The names are specified in upper case. The value for each item is then set using an equivalent lower case name in `SETNITEMC` [↗](#) (page 1712), but with the case forced to upper. The value is then obtained using `GETNITEMC`. The result is written to the log.

```
DATA _NULL_;

  ix = 0;

  lid1 = MAKENLIST('l', 'VAR1','VAR2','VAR3','VAR4');

  mm = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMC(lid1, 'bicycle','var1',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'plane', 'var2',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'horse', 'var3',,,ix,'y');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4',,,ix,'y');

  CALL PUTLIST(lid1,,0);

  rx = GETNITEMC(lid1,'var1',,,,'roller skate');
  PUT 'Returns: ' rx;

  rx = GETNITEMC(lid1,'VAR4',,,,'roller skate');
  PUT 'Returns: ' rx;

  rx = GETNITEMC(lid1,'var1',,,,'roller skate','y');
  PUT 'Returns: ' rx;

RUN;
```

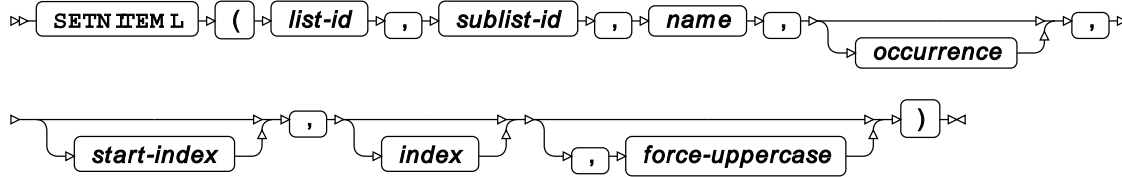
This produces the following output:

```
      (VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
)[1]
Returns: roller skate
Returns: taxi
Returns: bicycle
```

The first use of `GETNITEMC` returns the specified default value `roller skate` because the item name `var1` does not exist (`VAR1` does). The third use of `GETNITEMC` returns the value specified in `SETNITEMC` [↗](#) (page 1668) because the *force-uppercase* argument is set to `Y`.

SETNITEML

Sets the specified list item to the specified sublist.



The sublist specified overwrites the named item. It is not the sublist itself that is placed in the list, but a pointer to it, identified in other functions by the list identifier.

Return type: Numeric

The identifier of the list into which the sublist is inserted.

list-id

Type: List

sublist-id

Type: List

The identifier of the sublist that overwrites the item.

name

Type: Character

The name of the list item to be set as the sublist.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item to set. This must be greater than 0 (zero). The default is 1. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

index

Optional argument

Type: Var

The name of a variable in which the position of the item specified is returned. If the value of this argument is anything except a variable name, an error is returned.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, a new list is created that contains four named items. A second list is then created, containing three items. The second list is inserted into the first list. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  lr2 = SETNITEMN(lid2, 100, 'ov1');
  lr2 = SETNITEMN(lid2, 250, 'ov2');
  lr2 = SETNITEMN(lid2, 310, 'ov3');

  sir = SETNITEML(lid1, lid2, 'var2');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'  
  VAR2=(OV1=100  
    OV2=250  
    OV3=310  
  ) [2]  
  VAR3='horse'  
  VAR4='taxi'  
) [1]
```

The output shows that the sublist has been assigned to the specified item name. The value `horse`, originally specified for the named item `var2` in the first list, has now been overwritten with a pointer to the second list.

In this example, the case in which you specify the item name is ignored. You could, for example, have specified:

```
rc = SETNITEML(lid1, lid2, 'Var2');
```

The output would still be the same.

Example – including unspecified item names in the list

In this example, a new list is created that contains four named items. A second list is then created, containing three items. The second list is inserted into the first list, but at an item name that does not exist. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4');  
  
  lr1 = SETNITEMC(lid1, 'bicycle','var1');  
  lr1 = SETNITEMC(lid1, 'plane', 'var2');  
  lr1 = SETNITEMC(lid1, 'horse', 'var3');  
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');  
  
  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');  
  
  lr2 = SETNITEMN(lid2, 100, 'ov1');  
  lr2 = SETNITEMN(lid2, 250, 'ov2');  
  lr2 = SETNITEMN(lid2, 310, 'ov3');  
  
  sir = SETNITEML(lid1, lid2, 'var6');  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
var6=(OV1=100
      OV2=250
      OV3=310
      ) [2]
) [1]
```

The output shows that the sublist has been assigned to the specified item name, but as the name did not already exist it has been created as a new name in the list. On display, the case of the new item name matches the case with which it was specified.

Example – non-unique item names

In this example, a new list is created that can hold five named items, where two of the items have non-unique item names. A second list is then created, containing three items. The second list is inserted into the first list, but at a specified occurrence of the non-unique item name. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');
  lr1 = SETNITEMC(lid1, 'ship', 'var4',2);

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  lr2 = SETNITEMN(lid2, 100, 'ov1');
  lr2 = SETNITEMN(lid2, 250, 'ov2');
  lr2 = SETNITEMN(lid2, 310, 'ov3');

  sir = SETNITEML(lid1, lid2, 'var4', 2);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
var4=(OV1=100
      OV2=250
      OV3=310
      ) [2]
) [1]
```


The output shows that the sublist has been assigned to the specified occurrence of the item name. The value `ship`, originally specified for the named occurrence of the item `var4` in the first list, has now been overwritten with a pointer to the second list.

If you had not specified the occurrence to set, the first occurrence would be selected. For example:

```
rc = SETNITEML(lid1, lid2, 'var4');
```

produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4=(OV1=100
      OV2=250
      OV3=310
    ) [2]
VAR4='ship'
) [1]
```

The first occurrence of the item with the specified name is overwritten.

Example – starting from an index position

In this example, a new list is created that can hold six named items, where three of the items have non-unique item names. A second list is then created, containing three items. The second list is inserted into the first list, but at a specified occurrence of the non-unique item name. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4','var4', 'var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');
  lr1 = SETNITEMC(lid1, 'car', 'var4',2);
  lr1 = SETNITEMC(lid1, 'ship', 'var4',2,5);

  lid2 = MAKENLIST('l', 'ov1','ov2', 'ov3');

  lr2 = SETNITEMN(lid2, 100, 'ov1');
  lr2 = SETNITEMN(lid2, 250, 'ov2');
  lr2 = SETNITEMN(lid2, 310, 'ov3');

  sir = SETNITEML(lid1, lid2, 'var4',2,5);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'  
  VAR2='plane'  
  VAR3='horse'  
  VAR4='taxi'  
  VAR4='car'  
  VAR4=(OV1=100  
        OV2=250  
        OV3=310  
  ) [46]  
) [1]
```

The output shows that the list has been assigned to the specified item. The third occurrence of the item named `var4` has the list correctly assigned to it because the function searched for the item name starting from the fifth index position. The second occurrence from that starting position had the value `ship`, which has now been overwritten by the pointer to the sublist.

Example – specifying case

In this example, a new list is created that holds four named items. The names are specified in upper case. The value for each item is then set using an equivalent lower case name in `SETNITEMC`, but forced to upper case. The sublist is written to a specified named item, with the *force-uppercase* set to `Y`. This ensures that the item name in this function matches the case of the item name in the specified list. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('1', 'VAR1','VAR2','VAR3','VAR4');  
  
  mm = SETLATTR(lid1, 'HONORCASE');  
  
  lr1 = SETNITEMC(lid1, 'bicycle','var1',,,ix,'y');  
  lr1 = SETNITEMC(lid1, 'plane', 'var2',,,ix,'y');  
  lr1 = SETNITEMC(lid1, 'horse', 'var3',,,ix,'y');  
  lr1 = SETNITEMC(lid1, 'taxi', 'var4',,,ix,'y');  
  
  lid2 = MAKENLIST('1', 'ov1','ov2', 'ov3');  
  
  lr2 = SETNITEMN(lid2, 100, 'ov1');  
  lr2 = SETNITEMN(lid2, 250, 'ov2');  
  lr2 = SETNITEMN(lid2, 310, 'ov3');  
  
  sir = SETNITEML(lid1, lid2, 'var4',,,ix,'y');  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1='bicycle'
  VAR2='plane'
  VAR3='horse'
  VAR4=(OV1=100
        OV2=250
        OV3=310
        ) [2]
) [1]
```

The sublist has been correctly placed because the item name specified matches the case of the item name in the destination list. If you had not set *force-uppercase* to Y:

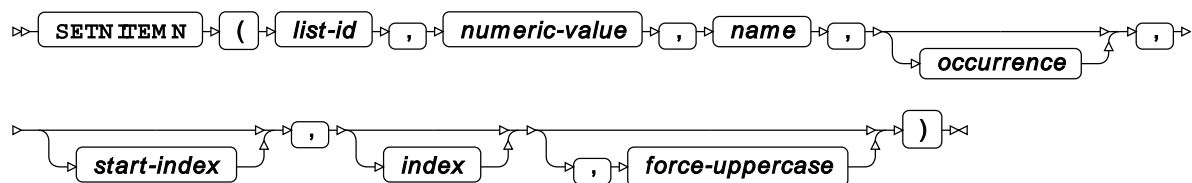
```
rc = SETNITEML(lid1, lid2, 'var4');
```

```
VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
var4=(OV1=100
      OV2=250
      OV3=310
      ) [2]
) [1]
```

then the sublist would be assigned to a new variable, `var4` rather than `VAR4` as case is being honoured as specified by setting the `HONORCASE` attribute of the list.

SETNITEMN

Sets the specified named list item to the specified number.



The number specified overwrites the current value of the specified named item.

If you want to insert a number without overwriting any items, see [INSERTN](#) (page 1654), which can be used to insert a number at a specified position in a list.

Return type: Numeric

The list identifier.

list-id

Type: List

The identifier for the list.

numeric-value

Type: Numeric

The value to which to set this list item. This must be a number.

name

Type: Character

The name of the list item for which you want to set a value.

occurrence

Optional argument

Type: Numeric

An integer that specifies the occurrence of the item to find, if it exists. This must be greater than 0 (zero). For example, to find the second occurrence of the item, specify 2. The default is 1. The search starts at the first item. If you want the search to start at another item, specify *start-index*.

start-index

Optional argument

Type: Numeric

An integer that specifies the ordinal position in the list at which to start searching for the named item to set. This must be greater than 0 (zero). The default is 1. This argument is useful when the item list contains non-unique item names, and you want to search for an occurrence after an index position.

index

Optional argument

Type: Var

The name of a variable in which the position of the item specified is returned. If the value of this argument is anything except a variable name, an error is returned.

force-uppercase

Optional argument

Defines whether an item name specified using lower case characters is treated as consisting entirely of upper case characters or not, or whether case is ignored. This option is only effective if the attribute 'HONORCASE' has been set for the list. Attributes are set using the [SETLATTR](#) (page 1706) function.

"N"

The item name is handled in its original case. This is the default.

"Y"

The item name is treated as if it consists of upper case characters.

Basic example

In this example, a new list is created that contains four named items. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  lr1 = SETNITEMN(lid1, 1,'var1');
  lr1 = SETNITEMN(lid1, 2, 'var2');
  lr1 = SETNITEMN(lid1, 3, 'var3');
  lr1 = SETNITEMN(lid1, 4, 'var4');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1='bicycle'
VAR2='plane'
VAR3='horse'
VAR4='taxi'
) [1]
```

The output shows that the values have been assigned to the specified values. The item names are displayed in upper case in the output.

In this example, the case in which you specify the item names is ignored. You could, for example, have specified:

```
lr1 = SETNITEMC(lid1, 'bicycle','VAR1');
lr1 = SETNITEMC(lid1, 'plane', 'var2');
lr1 = SETNITEMC(lid1, 'horse', 'Var3');
lr1 = SETNITEMC(lid1, 'taxi', 'var4');
```

The output would still be the same.

Example – including unspecified item names in the list

In this example, a new list is created that can hold four named items, but is extended to five items using `SETNITEMN` [\(page 1712\)](#). `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2','var3','var4');

  lr1 = SETNITEMC(lid1, 'bicycle','var1');
  lr1 = SETNITEMC(lid1, 'plane', 'var2');
  lr1 = SETNITEMC(lid1, 'horse', 'var3');
  lr1 = SETNITEMC(lid1, 'taxi', 'var4');
  lr1 = SETNITEMC(lid1, 'ship', 'var5');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1=1  
VAR2=2  
VAR3=3  
VAR4=4  
var5=7  
) [1]
```

The output shows that values have been assigned to the specified items, and a new item, named `var5`, has been added to the list. On display, the case of the new item name matches the case with which it was specified.

Example – non-unique item names

In this example, a new list is created that can hold five named items, where two of the items have non-unique item names. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4','var4');  
  
  lr1 = SETNITEMN(lid1, 1, 'var1');  
  lr1 = SETNITEMN(lid1, 2, 'var2');  
  lr1 = SETNITEMN(lid1, 3, 'var3');  
  lr1 = SETNITEMN(lid1, 4, 'var4');  
  lr1 = SETNITEMN(lid1, 100, 'var4',2);  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1=1  
VAR2=2  
VAR3=3  
VAR4=4  
VAR4=100  
) [1]
```

The output shows that values have been assigned to the specified items, and that the items with the same name have unique values because the occurrence for the item was specified. If the occurrence is not specified:

```
lr1 = SETNITEMN(lid1, 1, 'var1');  
lr1 = SETNITEMN(lid1, 2, 'var2');  
lr1 = SETNITEMN(lid1, 3, 'var3');  
lr1 = SETNITEMN(lid1, 4, 'var4');  
lr1 = SETNITEMN(lid1, 100, 'var4');
```

This produces the following output:

```
(VAR1=1  
VAR2=2  
VAR3=3  
VAR4=100  
VAR4=.  
) [1]
```

The first occurrence of the item with the specified name is overwritten, and the second occurrence of the item has a missing value.

In this example, three items with non-unique names are specified:

```
DATA _NULL_;  
  
  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4', 'var4', 'var4');  
  
  lr1 = SETNITEMN(lid1, 1, 'var1');  
  lr1 = SETNITEMN(lid1, 2, 'var2');  
  lr1 = SETNITEMN(lid1, 3, 'var3');  
  lr1 = SETNITEMN(lid1, 4, 'var4');  
  lr1 = SETNITEMN(lid1, 100, 'var4',2);  
  lr1 = SETNITEMN(lid1, 50, 'var4');  
  
  CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(VAR1=1  
VAR2=2  
VAR3=3  
VAR4=50  
VAR4=100  
VAR4=.  
) [1]
```

The first occurrence of the non-unique item name is overwritten, the second occurrence of the item name is set correctly because *occurrence* has been specified, and the third occurrence is set to missing value.

Example – starting from an index position

In this example, a new list is created that can hold six named items, where three of the items have non-unique item names. `CALL PUTLIST` is used to write the list to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4', 'var4', 'var4');

  lr1 = SETNITEMN(lid1, 1, 'var1');
  lr1 = SETNITEMN(lid1, 2, 'var2');
  lr1 = SETNITEMN(lid1, 3, 'var3');
  lr1 = SETNITEMN(lid1, 4, 'var4');
  lr1 = SETNITEMN(lid1, 100, 'var4',2);
  lr1 = SETNITEMN(lid1, 50, 'var4',2,5);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(VAR1=1
VAR2=2
VAR3=3
VAR4=4
VAR4=100
VAR4=50
) [1]
```

The output shows that values have been assigned to the specified items. The third occurrence of the item named `var4` has the correct value assigned because the function searched for the item name starting from the fifth index position. The second occurrence from that starting position has the value 50.

Example – specifying case

In this example, `MAKENLIST` [↗](#) (page 1668) is used to create a new list that consists of four named items. The names are specified in upper case. The value for each item is then set using an equivalent lower case name in `SETNITEMN` [↗](#) (page 1725). The value is then obtained using `GETNITEMN`. The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKENLIST('l', 'var1','var2', 'var3', 'var4');

  mm = SETLATTR(lid1, 'HONORCASE');

  lr1 = SETNITEMN(lid1, 100, 'var1',,,ix,'y');
  lr1 = SETNITEMN(lid1, 250, 'var2',,,ix,'y');
  lr1 = SETNITEMN(lid1, 310, 'var3',,,ix,'y');
  lr1 = SETNITEMN(lid1, 40,  'var4',,,ix,'y');

  CALL PUTLIST(lid1,,0);

  gi = GETNITEMN(lid1,'var1',,, 0);
  PUT 'Returns: ' gi;

  gi = GETNITEMN(lid1,'VAR4',,, 0, 'y');
  PUT 'Returns: ' gi;

  gi = GETNITEMN(lid1,'var1',,, 0, 'y');
  PUT 'Returns: ' gi;

RUN;
```

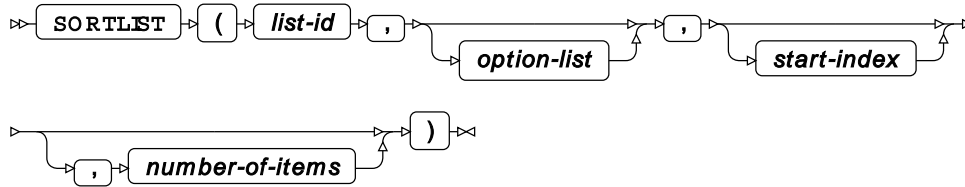
This produces the following output:

```
(VAR1=100
VAR2=250
VAR3=310
VAR4=40
)[1]Returns: 0
Returns: 40
Returns: 100
```

The first use of `GETNITEMN` returns the specified default value 0 because the item name `var1` does not exist (`VAR1` does). The third use of `GETNITEMN` returns the value specified in `SETNITEMN` because the *force-uppercase* argument is set to `Y`.

SORTLIST

Sorts the specified list.



Return type: Numeric

list-id

Type: List

The identifier for the list.

option-list

Optional argument

Specifies how the sort should proceed; for example, ascending or descending, taking into account case or not, and so on. Options should be separated by space, and the entire list should be quoted, not each option; for example, 'DESCENDING OBEYCASE'. The following options are available:

"ASCENDING"

Sort the list in ascending order. This is the default.

"DESCENDING"

Sort the list in descending order.

"IGNORECASE"

Ignore the case of character items. This is the default.

"OBEYCASE"

Sort obeying rules of case.

"VALUE"

For named list items, sort according to the value of named items.

"NAME"

For named list items, sort according to the names of the items.

"NODUP"

Duplicates are ignored, and dropped from the list.

start-index

Optional argument

Type: Numeric

The ordinal position at which to start the sort. If not specified, sorting starts at the first position. If *number-of-items* is not specified, sorting occurs from the starting point to the end of the list.

number-of-items

Optional argument

Type: Numeric

The number of items to be sorted from the starting position, which is either *start-index*, or the first item if *start-index* is not specified.

Basic example

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMN` [\(page 1703\)](#) functions. The items are then extracted from the list using `GETITEMN` [\(page 1631\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(4);

  rc = SETITEMN(lid1, 100, 1);
  rc = SETITEMN(lid1, 600, 2);
  rc = SETITEMN(lid1, 450, 3);
  rc = SETITEMN(lid1, 200, 4);

  rc = SORTLIST(lid1);

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
(100
 200
 450
 600
) [1]
```

Example – sort with options

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. The items are then extracted from the list using `GETITEMC` [\(page 1629\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(6);

  lr1 = SETITEMC(lid1, 'car', 1);
  lr1 = SETITEMC(lid1, 'bicycle', 2);
  lr1 = SETITEMC(lid1, 'train', 3);
  lr1 = SETITEMC(lid1, 'Bicycle', 4);
  lr1 = SETITEMC(lid1, 'Bicycle', 5);
  lr1 = SETITEMC(lid1, 'Plane', 6);

  slr = SORTLIST(lid1, 'DESCENDING OBEYCASE NODUP');

  CALL PUTLIST(lid1,,0);

RUN;
```

This produces the following output:

```
('train'
'car'
'bicycle'
'Plane'
'Bicycle'
)[1]
```

In this example, the list has been sorted into descending order. Case is taken into account, and as upper case characters are lower in the collating order than lower case characters, strings containing upper case characters appear at the end of the list. Duplicates are not allowed, so one of the instances of the string `Bicycle` has been removed from the list and the resulting list contains five items rather than six.

Example – sort with start and end points

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMN` [\(page 1703\)](#) functions. The items to be sorted start at the specified item, and include the specified number of items. Because no options have been specified, *option-list* is null but its delimiter (,) is included in the function. The items are then extracted from the list using `GETITEMN` [\(page 1631\)](#). The result is written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(16);

  lr1 = SETITEMN(lid1, 100, 1);
  lr1 = SETITEMN(lid1, 600, 2);
  lr1 = SETITEMN(lid1, 450, 3);
  lr1 = SETITEMN(lid1, 200, 4);
  lr1 = SETITEMN(lid1, 250, 5);
```

```
lr1 = SETITEMN(lid1, 700, 6);  
lr1 = SETITEMN(lid1, 150, 7);  
lr1 = SETITEMN(lid1, 220, 8);  
lr1 = SETITEMN(lid1, 100, 9);  
lr1 = SETITEMN(lid1, 600, 10);  
lr1 = SETITEMN(lid1, 450, 11);  
lr1 = SETITEMN(lid1, 210, 12);  
lr1 = SETITEMN(lid1, 250, 13);  
lr1 = SETITEMN(lid1, 760, 14);  
lr1 = SETITEMN(lid1, 160, 15);  
lr1 = SETITEMN(lid1, 230, 16);  
  
slr = SORTLIST(lid1,,4,10);  
  
CALL PUTLIST(lid1,,0);  
  
RUN;
```

This produces the following output:

```
(100  
 600  
 450  
 100  
 150  
 200  
 210  
 220  
 250  
 250  
 450  
 600  
 700  
 760  
 160  
 230  
) [1]
```

In this example, the list has been sorted into ascending order starting from the fourth character and continuing for a further ten characters.

CALL LISTPROF

Enables the number of lists created in a `DATA` step to be calculated.

```
⇒ CALL LISTPROF ⇒ ( ⇒ profiler ⇒ ) ⇒ ; ⇒
```

This routine should be called before `GETLCNTP` [↗](#) (page 1635), which returns the number of lists created during the `DATA` step.

profiler

"ON"

Switches the list profiler on.

"OFF"

Switches the list profiler off. This is the default.

The number of lists created between `CALL LISTPROF('ON')` and `CALL LISTPROF('OFF')` is calculated. The count is restarted when `CALL LISTPROF('ON')` is next used.

Basic example

In this example, two lists are created. One list is deleted, and then another created. This is done three times. The result is written to the log.

```
DATA _NULL_;

CALL LISTPROF('ON');

DO i = 1 TO 3;
  lid1 = MAKENLIST('l', 'v1', 'v2', 'v3');
  lid2 = MAKENLIST('l', 'v1', 'v2', 'v3');

  rc = DELLIST(lid2);

  lid3 = MAKENLIST('l', 'var1', 'var2', 'var3');
END;

rp = GETLCNTP();
PUT 'Number of lists created: ' rp;

RUN;
```

This produces the following output:

```
Number of lists created: 6
```

The `DATA` step creates nine lists, but also deletes three.

Example – showing counts as a result of using `ON` and `OFF`

In this example, various lists are created. The result is written to the log.

```
DATA _NULL_;

CALL LISTPROF('ON');

lid1 = MAKENLIST('l', 'v1', 'v2', 'v3');
lid2 = MAKENLIST('l', 'v2', 'v3', 'v4');

rp = GETLCNTP();

PUT 'Number of lists created: ' rp;

CALL LISTPROF('OFF');
```

```

lid3 = MAKENLIST('l', 'v5', 'v6', 'v7');
lid4 = MAKENLIST('l', 'v8', 'v9', 'v10');

rp = GETLCNTP();

PUT 'Number of lists created: ' rp;

CALL LISTPROF('OFF');

lid5 = MAKENLIST('l', 'v1', 'v2', 'v3');
lid6 = MAKENLIST('l', 'v2', 'v3', 'v4');

rp = GETLCNTP();

PUT 'Number of lists created: ' rp;

lid7 = MAKENLIST('l', 'n1', 'n2', 'n3');
lid8 = MAKENLIST('l', 'n4', 'n5', 'n6');

rp = GETLCNTP();

PUT 'Number of lists created: ' rp;

CALL LISTPROF('OFF');

RUN;

```

This produces the following output:

```

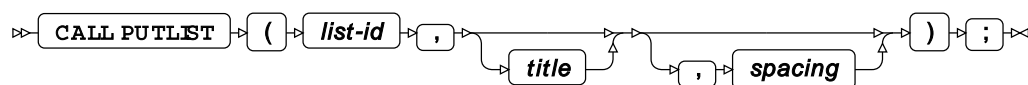
Number of lists created: 2
Number of lists created: 2
Number of lists created: 2
Number of lists created: 4

```

Eight lists are created in the DATA step. `CALL LISTPROF('ON')` starts the count of lists. Two lists are created, then `GETLCNTP` is used to return the number of lists created; 2 is returned. `CALL LISTPROF('OFF')` is then used to switch off counting. Two more lists are created, and then `GETLCNTP` is used to return the number of lists created. Again, 2 is returned, because counting was switched off. `CALL LISTPROF('ON')` is then used again to restart counting. Two more lists are created, and then `GETLCNTP` is used to return the number of lists created; 2 is returned. Two more lists are then created, and then `GETLCNTP` is used to return the number of lists created; 4 is returned, as four lists were created (and none deleted) between the `CALL LISTPROF('ON')` and `CALL LISTPROF('ON')`.

CALL PUTLIST

Write the specified list to the log.



list-id**Type:** List

The identifier for the list.

title

Optional argument

Type: Character

The title for the list.

spacing

Optional argument

Type: Numeric

Lay out the list vertically, and set the indent. The indent is applied to each item in the list. The value for this argument can be any positive integer, including 0 (zero). By default, the list is laid out horizontally.

The list written to the log has one of the following formats:

```
('item' [, 'item', ...]) [n]
```

where *item* is a list item, and *n* is the list identifier. If the list contains named items, it will have this format:

```
(name='item' [, name=item, ...]) [n]
```

where *name* is the name of an item inserted with the `SETNITEMC`, `SETITEMN` or `SETITEML` functions into a list created with `MAKENLIST`.

If you specify a value for *title*, the list has the following format:

```
title={list} [n]
```

where *title* is the title of the list, and *list* is the list in one of the preceding formats.

The same rules apply if the list is laid out vertically, except that the values are indented, and any sublists are indented. The title, if specified, and the opening parenthesis fix the left margin of the vertical layout; the number of any indent characters, as specified by *spacing*, are counted from that margin. For example:

```
CALL PUTLIST(lid1,'List',2);
```

creates the output:

```
List=(  'car'  
      'bicycle'  
      'train'  
      ) [37]
```


In this example, the first list item is positioned two characters from the opening parenthesis. All subsequent list items are positioned in line with the first item. Items in sublists will be positioned in the same way. Sublists do not have a title, but a name can be specified for the sublist when it is inserted into a list, and this name has the same layout in the log as a title has.

If you insert a sublist into a list, the list has one of the formats above, and is positioned in the output at the index position it has in the list. If the output is displayed vertically, the sublist is indented.

Basic example

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMC](#) (page 1696) functions. The list is then written to the log.

```
DATA _NULL_;

  lid1 = MAKELIST(3);

  lr1= SETITEMC(lid1, 'car', 1);
  lr1= SETITEMC(lid1, 'bicycle', 2);
  lr1= SETITEMC(lid1, 'train', 3);

  CALL PUTLIST(lid1);

RUN;
```

This produces the following output:

```
('car' 'bicycle' 'train' ) [1]
```

Example – list with title

In this example, a list is created using the [MAKELIST](#) (page 1667) and [SETITEMC](#) (page 1696) functions. The list is then written to the log with the title specified.

```
DATA _NULL_;

  lid1 = MAKELIST(3);

  lr1= SETITEMC(lid1, 'car', 1);
  lr1= SETITEMC(lid1, 'bicycle', 2);
  lr1= SETITEMC(lid1, 'train', 3);

  CALL PUTLIST(lid1, 'Transport');

RUN;
```

This produces the following output:

```
Transport=('car' 'bicycle' 'train' ) [2]
```

Example – list with title and vertical layout

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. The list is then written to the log with a specified title, and laid out vertically.

```
DATA _NULL_;

  lid1 = MAKELIST(3);

  lr1= SETITEMC(lid1, 'car', 1);
  lr1= SETITEMC(lid1, 'bicycle', 2);
  lr1= SETITEMC(lid1, 'train', 3);

  CALL PUTLIST(lid1, 'Transport',0);

RUN;
```

This produces the following output:

```
Transport=('car'
          'bicycle'
          'train'
          ) [1]
```

Here, 0 (zero) has been specified to *spacing*, so the list is displayed vertically, but no indentation is applied.

Example – list with title, vertical layout, indentation, item names and sublist

In this example, a list is created using the `MAKELIST` [\(page 1667\)](#) and `SETITEMC` [\(page 1696\)](#) functions. Another list is similarly created, and this list is then inserted into the first list as a sublist. The first list is then written to the log with the title specified, and with an indented vertical layout.

```
DATA _NULL_;

  lid1 = MAKENLIST('g', 'var1', 'var2', 'var3', 'var4');

  lr1= SETNITEMC(lid1, 'ship', 'var1');
  lr1= SETNITEMC(lid1, 'plane', 'var2');
  lr1= SETNITEMC(lid1, 'horse', 'var3');
  lr1= SETNITEMC(lid1, 'taxi', 'var4');

  lid2 = MAKENLIST('g', 'lst1', 'lst2', 'lst3');

  lr2= SETNITEMC(lid2, 'car', 'lst1');
  lr2= SETNITEMC(lid2, 'train', 'lst2');
  lr2= SETNITEMC(lid2, 'bicycle', 'lst3');

  ir = INSERTL(lid1, lid2, 2, 'Sub1');

  CALL PUTLIST(lid1, 'Transport',5);

RUN;
```

This produces the following output:

```
Transport=(      VAR1='ship'
                Sub1=(      LST1='car'
                            LST2='train'
                            LST3='bicycle'
                        ) [2]
                VAR2='plane'
                VAR3='horse'
                VAR4='taxi'
            ) [1]
```

In this example, the value 5 has been specified for *spacing*, so the list is displayed vertically and a five character indent is applied. The sublist is positioned at the index position specified by the [INSERTL](#) (page 1656) function, with the name also specified in that function. The sublist is indented within the list, using the same indentation.

ISPF CALL routines

Use the ISPF service to perform tasks.

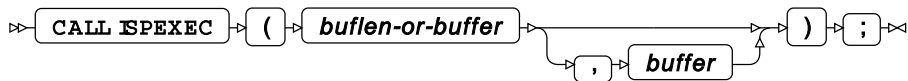
The Interactive System Productivity Facility (ISPF) is a service available on z/OS on IBM mainframes. The service provides IBM 3270 terminals with an interface through which users can perform tasks on the mainframe.

For details on ISPF, see your IBM documentation.

CALL ISPEXEC	1741
Execute ISPF services. Parameters are specified in a buffer.	
CALL ISPLINK	1742
Execute ISPF services by specifying parameters in the routine.	

CALL ISPEXEC

Execute ISPF services. Parameters are specified in a buffer.



This function enables you to execute service requests by specifying parameters in a buffer, which you then specify in the routine.

If you need to pass the values of variables between the calling program and ISPF, you must use [CALL ISPLINK](#) (page 1742).

buflen-or-buffer

Type: Character or numeric value

If *buffer* is not specified, this argument specifies the buffer containing the name of the services and parameters. The length of the buffer is calculated by the routine.

If *buffer* is specified, this must be the length of the buffer.

buffer

Optional argument

Type: Character

A buffer containing the name of the services and parameters. The length of the buffer is specified by *buflen-or-buffer*. For information on the parameters that can be specified, see your IBM ISPF documentation

This function implements the call format of the ISPF `ISPEXEC` interface. For detailed information on `ISPEXEC`, `CALL ISPEXEC` and ISPF, see your IBM ISPF documentation. Also see that documentation for detailed examples of using the routine.

Example

In this example, the DISPLAY interface is started with the specified panel name.

```
DATA _NULL_;  
  buflen = 16;  
  buffer = 'DISPLAY PANELID1';  
  CALL ISPEXEC(buflen, buffer);  
RUN;
```

This example will do nothing outside the context of a corresponding ISPF environment, and is only provided to show the form of the routine.

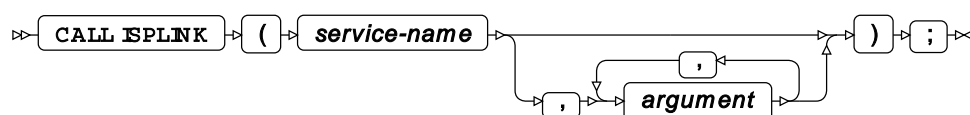
The routine in this example could also have been specified as:

```
CALL ISPEXEC(buffer);
```

The length of the buffer would then calculated by the function.

CALL ISPLINK

Execute ISPF services by specifying parameters in the routine.



This function enables you to execute service requests by specifying parameters in the call. If you need to pass the values of variables between the calling program and ISPF, you must use this routine.

service-name

Type: Character

The name of an ISPF service.

argument

Optional argument

Type: Character or numeric value

An argument or parameter required by the service.

Arguments are positional. Optional arguments on the right-hand side of the argument list can be omitted if not required. If you want to omit an argument within the list, specify it as a blank enclosed in quotes (' ').

This function implements the call format of the ISPF `ISPLINK` interface. For detailed information on `ISPLINK`, `CALL ISPLINK` and ISPF, see your IBM ISPF documentation. Also see that documentation for detailed examples of using the routine.

Example

In this example, the `DISPLAY` interface is started with the specified panel.

```
DATA _NULL_;
  LENGTH NAME $40 TITLE $20;
  CALL ISPLINK('VDEFINE', 'NAME TITLE');
  CALL ISPLINK('DISPLAY', 'PanelID1');
RUN;
```

This example will do nothing outside the context of a corresponding ISPF environment, and is only provided to show the form of the routine.

Macro functions and `CALL` routines

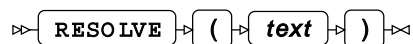
Manipulate macro variables and execute macros in the `DATA` step.

<code>RESOLVE</code> ↗	1744
Returns the result of a macro expression, which can then be used in the <code>DATA</code> step.	
<code>SYMEXIST</code> ↗	1745
Returns a value indicating whether a specified macro variable exists.	
<code>SYMGET</code> ↗	1746
Returns the value of a macro variable as character data.	
<code>SYMGETN</code> ↗	1747
Returns the value of a macro variable as numeric data.	
<code>SYMGLOBL</code> ↗	1748
Returns a value that indicates whether a macro variable is global.	

SYMLOCAL ↗	1749
Returns a value that indicates whether a macro variable is local.	
CALL EXECUTE ↗	1750
Enables code, such as DATA steps, procedures and macros, to be run from a DATA step.	
CALL SYMDEL ↗	1752
Deletes a global variable name and its value.	
CALL SYMPUT ↗	1753
Assigns a character value to a global macro variable that can subsequently be used in a macro.	
CALL SYMPUTN ↗	1754
Assigns a numeric value to a global macro variable that can be subsequently used in a macro.	
CALL SYMPUTX ↗	1755
Assigns a value to a global or local macro variable that can be subsequently used in a macro.	

RESOLVE

Returns the result of a macro expression, which can then be used in the DATA step.



Return type: Character

text

Type: Character

A macro expression.

Basic example

In this example, a macro variable is resolved and the value provided to the DATA step. The result is written to the log.

```

%MACRO an;
  Marwick, Arthur
%MEND an;

LIBNAME books "c:\temp\books";
DATA _NULL_;
  retain chk 0;
  SET books.books;

  IF author EQ resolve('%an') AND chk NE 1 THEN do;
    chk = 1;
    name = quote(resolve('%an'));
    PUT "Author " name " exists";
  end;
run;

```

This produces the following output:

```
Author "Marwick, Arthur"  exists
```

Example – executing macro in DATA step

In this example, a macro variable is resolved in the DATA step, and statements specified as a macro are executed. The result is written to the `example` dataset.

```
%MACRO an;
  Hist
%MEND an;

LIBNAME books "c:\temp\books";
DATA example;
  SET books.books;

  IF LENGTH(type) >= 4 THEN
    DO;
      IF RESOLVE('%SUBSTR(' || TYPE || ', 1, 4)') = RESOLVE('%an')
        THEN OUTPUT;
    END;
  RUN;
```

The dataset `example` contains all observations in which the variable `Type` begins with the text `Hist` (such as `History`, `Hist of Science`, `Hist/Culture` and so on).

Note:

This example would return the same results if the `SUBSTR` function was used in the DATA step, rather than as a statement in the macro in the `RESOLVE` statement; for example:

```
IF SUBSTR(TYPE, 1, 4) = RESOLVE('%an')
```

SYMEXIST

Returns a value indicating whether a specified macro variable exists.

➤ **SYMEXIST** ➤ (➤ *symbol-name* ➤) ➤

Before creating a macro variable, you might want to check whether the variable name already exists in the global or local symbol tables. This function enables you to do that.

Return type: Numeric

- 0 The named variable does not exist
- 1 The named variable exists

symbol-name

Type: Character

The name of the macro variable.

Example

In this example, the function is used to check whether a named variable exists. The result is written to the log.

```
%LET an = 0;

DATA _NULL_;
  se = SYMEXIST("an");
  se = IFC(se, "The variable exists", "The variable doesn't exist");
  PUT se;
RUN;
```

This produces the following output:

```
The variable exists
```

The variable is first set using a %LET statement; the subsequent DATA step checks whether the named variable exists.

SYMGET

Returns the value of a macro variable as character data.

⇒ **SYMGET** ⇒ (⇒ *symbol-name* ⇒) ⇒

Return type: Character

symbol-name

Type: Character

The name of the macro variable containing the value to be obtained.

If the macro variable contains a numeric, this will be returned as character data.

Example

In this example, the function is used to get a character value from a macro variable. The result is written to the log.

```
%LET an = House;
DATA _NULL_;

  gmv = SYMGET("an");
  PUT "The value of the symbol is: " gmv;

RUN;
```


This produces the following output:

```
The value of the symbol is: House
```

The variable is first set using a %LET statement; the subsequent DATA step gets the value and writes it in the log.

SYMGETN

Returns the value of a macro variable as numeric data.

⇒ **SYMGETN** ⇒ (⇒ *macro-variable* ⇒) ⇒

A value specified to a macro variable is assigned to the variable as text. This function ensures that a variable containing a number is returned as numeric data, rather than as character data. If the variable specified to this function does not contain numeric data, an error is returned.

Return type: Numeric

macro-variable

Type: Character

The name of the macro variable containing the value to be obtained.

Example

In this example, the function is used to get a numeric value from a macro variable. The result is written to the log.

```
%LET an = 100;
DATA _NULL_;

    gmv = SYMGETN("an");
    PUT "The value of the symbol is: " gmv;

RUN;
```

This produces the following output:

```
The value of the symbol is: 100
```

The variable is first set using a %LET statement; the subsequent DATA step gets the value and writes it in the log.

SYMGLOBL

Returns a value that indicates whether a macro variable is global.

➤ SYMGLOBL (*symbol-name*) ➤

Return type: Numeric

- 0 The named variable is not a global variable.
- 1 The named variable is a global variable.

symbol-name

Type: Character

The name of the macro variable.

Example

In this example, the function is used to check whether the specified macro variables are local. The result is written to the log.

```
%LET yy = somedata;
%MACRO test;
  DATA _NULL_;
    %LET xx = moredata;

    ly = IFC(SYMGLOBL("xx"), "The variable xx is global", "The variable xx is not
global");
    PUT ly;

    ly = IFC(SYMGLOBL("yy"), "The variable yy is global", "The variable yy is not
global");
    PUT ly;

  RUN;
%MEND test;

%test;
```

This produces the following output:

```
The variable xx is not global
The variable yy is global
```

The variable `yy` is set using a `%LET` statement outside of the macro, and is therefore global. The variable `xx` is set using a `%LET` statement inside the macro, and is therefore local.

SYMLOCAL

Returns a value that indicates whether a macro variable is local.

➤ **SYMLOCAL** ➤ (➤ *symbol-name* ➤) ➤

Return type: Numeric

- 0 The named variable is not a global variable.
- 1 The named variable is a global variable.

symbol-name

Type: Character

The name of the macro variable.

Example

In this example, the function is used to check whether the specified macro variables are local. The result is written to the log.

```
%LET yy = somedata;
%MACRO test;
  DATA _NULL_;
    %LET xx = moredata;

    ly = IFC(SYMLOCAL("xx"), "The variable xx is local", "The variable xx is not
local");
    PUT ly;

    ly = IFC(SYMLOCAL("yy"), "The variable yy is local", "The variable yy is not
local");
    PUT ly;

  RUN;
%MEND test;

%test;
```

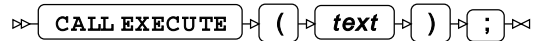
This produces the following output:

```
The variable xx is local
The variable yy is not local
```

The variable `yy` is set using a `%LET` statement outside of the macro, and is therefore global. The variable `xx` is set using a `%LET` statement inside the macro, and is therefore local.

CALL EXECUTE

Enables code, such as `DATA` steps, procedures and macros, to be run from a `DATA` step.



The code is provided as an argument. The code is run immediately after the `DATA` step that contains the `CALL` routine has finished. If the `DATA` step contains more than one `CALL EXECUTE`, the code contained in the routines is executed in the order in which the routines are specified. If a `CALL EXECUTE` itself contains a `CALL EXECUTE`, the code in the contained routine executes after the containing `CALL EXECUTE` has finished.

text

Type: Character

The text that comprises the code to be run.

The code can contain variables set in the containing `DATA` step, and functions that operate on variables. Variables and statements are identified and delimited by `||` (double vertical bars or pipes) in *text*. For example:

```
CALL EXECUTE("proc print data=" || filename || ";" )
```

contains the variable `filename`. In this example:

```
CALL EXECUTE("proc print data=" || trim(filename) || ";" )
```

the `TRIM` function is used to remove any leading or trailing spaces from the content of the variable `filename`.

Basic example

In this example, the `CALL` routine is used to run a `DATA` step. The result is written to the log.

```
DATA _NULL_;

  CALL EXECUTE("DATA _NULL_; PUT 'DATA step run from DATA step'; RUN;");

  PUT 'The called DATA step will run after this step';

RUN;
```

This produces the following output:

```
The called DATA step will run after this step

DATA step run from DATA step
```

Intervening output in the log has been removed. The called `DATA` step has been run after the containing `DATA` step.

Example – running a procedure

In this example, the routine is used to run a procedure. The result is written to the ODS output.

```
LIBNAME books 'c:\temp\books';
DATA _NULL_;
    filename = "books.books";
    CALL EXECUTE("PROC PRINT DATA=" || filename || ";");
RUN;
```

In this example, PROC PRINT is used to write the contents of the dataset books to the ODS output. The dataset name has been assigned to a variable which is then used in CALL EXECUTE to specify the dataset.

Example – including a macro

In this example, the routine is used run a macro. The result is written to the ODS output.

```
%MACRO an;
    PROC PRINT DATA=books.books;
        TITLE "A Book List";
%MEND an;

LIBNAME BOOKS "c:\temp\books";
DATA _NULL_;
    filename = "books.books";
    CALL EXECUTE('%an');
```

In this example, the macro contains the PRINT procedure, which writes the contents of the specified dataset, if it exists, to the ODS output.

Example – including a macro and statements

In this example, the routine is used to run a procedure. A macro is also created and included in the routine. The result is written to the ODS output.

```
%MACRO title;
    A Book List
%MEND title;

LIBNAME BOOKS 'c:\temp\books';
DATA _NULL_;

    filename = "books.books    ";
    CALL EXECUTE("PROC PRINT DATA=" || TRIM(filename) || ";TITLE %title;");

RUN;
```

In this example, PROC PRINT is used to write the contents of the dataset books to the ODS output. The dataset name has been assigned to a variable which is then used in CALL EXECUTE to specify the dataset. The macro specifies the title to be used in the procedure. The TRIM statement is used in the routine to remove any spaces from the filename.

Example – hierarchy of invocations

In this example, the routine is used five times, including an instance where the routine calls another `CALL EXECUTE` routine. The result is written to the log.

```
DATA _NULL_;

  CALL EXECUTE("DATA _NULL_; PUT 'Included DATA step 1'; RUN;");

  PUT 'The called DATA steps will run after this containing step';

  CALL EXECUTE("DATA _NULL_; PUT 'Included DATA step 2'; RUN;");

  CALL EXECUTE("DATA _NULL_;
    CALL EXECUTE('DATA _NULL_; PUT " || quote("Included DATA step 4") || "; RUN'; PUT
    'Included DATA step 3';RUN;');
    RUN;");

  CALL EXECUTE("DATA _NULL_; PUT 'Included DATA step 5'; RUN;");

RUN;
```

This produces the following output:

```
The called DATA steps will run after this containing step

Included DATA step 1

Included DATA step 2

Included DATA step 3

Included DATA step 4

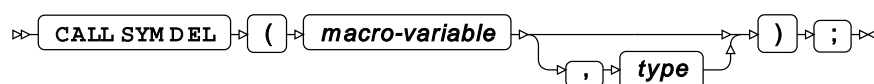
Included DATA step 5
```

Intervening output in the log has been removed. The called `DATA` step has been run after the containing `DATA` step.

In this example, a `CALL EXECUTE` routine has been called by another `CALL EXECUTE` routine. The text to be written to the log in the called routine has been created using a function entered within the routine using escape characters. This was required as the pairing of quote marks necessary to include a quoted string was too complex.

CALL SYMDEL

Deletes a global variable name and its value.



macro-variable

Type: Character

The name of the macro variable.

type

Optional argument

"NOWARN"

If the specified variable does not exist, or is not a global variable, no warning is written to the log.

Example

In this example, the function is used to delete two global variables. The result is written to the log.

```
%LET zz = 180;  
DATA _NULL_;  
    CALL SYMDEL("xx");  
    CALL SYMDEL("zz");  
RUN;
```

This produces the following output:

```
WARNING: Macro variable xx does not exist, or is not a global macro variable
```

The global variable `zz` exists, so it is deleted and no warning is generated. The global variable `xx` is a local variable, so a warning is written to the log.

CALL SYMPUT

Assigns a character value to a global macro variable that can subsequently be used in a macro.

⇒ **CALL SYMPUT** ⇒ (⇒ **variable-name** ⇒ , ⇒ **value** ⇒) ⇒ ; ⇒

This function enables you to assign a value to a variable in the `DATA` step.

variable-name

Type: Character

The name of the macro variable to which a value is assigned.

value

Type: Character

The value to assign to the macro variable.

Example

In this example, the function is used to assign data to a macro variable. The variable `vv` is then used in a macro. The result is written to the log.

```
data _null_;  
    call symput("vv", "Elephant");  
run;  
%macro test;  
    %put The value of the variable is &vv;  
%mend test;  
  
%test;
```

This produces the following output:

```
The value of the variable is Elephant
```

CALL SYMPUTN

Assigns a numeric value to a global macro variable that can be subsequently used in a macro.

→ **CALL SYMPUTN** → (→ *variable-name* → , → *value* →) → ; →

This function enables you to assign a value to a variable in the `DATA` step.

variable-name

Type: Character

The name of the macro variable to which a value is assigned.

value

Type: Numeric

The value to assign to the macro variable.

This must be a numeric value. If you specify a quoted string an error occurs and a message is written to the log.

Example

In this example, the function is used to assign data to a macro variable. The variable `vv` is then used in a macro. The result is written to the log.

```
DATA _NULL_;

    CALL SYMPUTN("vv", 100);
RUN;

%MACRO test;
    %PUT The value of the variable is &vv;
%MEND test;

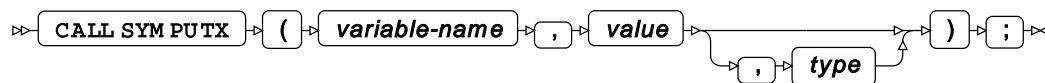
%TEST;
```

This produces the following output:

```
The value of the variable is 100
```

CALL SYMPUTX

Assigns a value to a global or local macro variable that can be subsequently used in a macro.



This function enables you to assign a value to a variable in the `DATA` step.

This function also removes any leading and trailing spaces from the specified value.

variable-name

Type: Character

The name of the macro variable to which a value is assigned.

value

Type: Character or numeric value

The value to assign to the macro variable.

This can be a numeric value or a quoted string.

type

Optional argument

Specifies whether the variable is treated as a global or local variable, or as a variable specified using `%LET`.

"G"

The variable is created as a global.

"F"

The variable is created as local.

"L"

The variable is created as global or local, according to the current scope, if no variable with that name already exists.

Example

In this example, the function is used to assign data to macro variables in a DATA step, and to specify what scope the variables have. The variables are then used in a macro. The result is written to the log.

```
%macro test2;
  data _null_;

    call symputx("vc", "cheese", "l");
    call symputx("vb", "bread", "g");
    call symputx("vy", "yellowhammer", "F");

    yo = symglobl("vc");
    msg = ifc(yo,"a global", "a local");
    put "vc is " msg;

    yo = symglobl("vb");
    msg = ifc(yo,"a global", "a local");
    put "vb is " msg;

    yo = symglobl("vy");
    msg = ifc(yo, "a global", "a local");
    put "vy is " msg;

run;
%mend test2;

%test2;

%macro test;

  %put The value of the variable is &vc;
  %put The value of the variable is &vb;
  %put The value of the variable is &vy;

%mend test;

%test;
```

This produces the following output:

```
vc is a local
vb is a global
vy is a local

The value of the variable is cheese
The value of the variable is bread
The value of the variable is yellowhammer
```

Mathematical functions and CALL routines

Perform mathematical operations on the data.

Constant functions ↗	1758
Define mathematical constants and numerical limits.	
Arithmetic functions ↗	1769
Perform arithmetic operations on the data.	
Power and exponent functions ↗	1777
Perform exponential operations on the data.	
Logarithmic functions ↗	1783
Perform logarithmic operations on the data.	
Trigonometric functions ↗	1788
Perform trigonometric operations on the data.	
Factorials and special functions ↗	1809
Perform factorial operations on the data and calculate special functions: Beta, Gamma, Bessel, Airy, Error and related functions.	
Value counts ↗	1832
Return counts for lists of values.	
Minimum and maximum values ↗	1834
Operate on the minimum and maximum in a list of values.	
Percentile-based calculations ↗	1840
Calculate percentile-based values.	
Sums and sums of squares ↗	1855
Calculate sums and sums of squares of a list of numeric values.	
Mean calculations ↗	1862
Calculate mean values.	
Variance, skewness and kurtosis calculations ↗	1869
Calculate functions based on the moments about the mean: variance, skewness, kurtosis and related.	

Constant functions

Define mathematical constants and numerical limits.

Mathematical constants ↗	1758
Return mathematical constants.	
Numerical limits ↗	1760
Return platform-dependent numerical limits.	

Mathematical constants

Return mathematical constants.

CONSTANT – E ↗	1758
Returns the mathematical constant $e \approx 2.718281828459$.	
CONSTANT – EULER ↗	1759
Returns the Euler constant $\gamma \approx 0.577215664902$.	
CONSTANT – PI ↗	1759
Returns the mathematical constant $\pi \approx 3.141592653690$.	

CONSTANT – E

Returns the mathematical constant $e \approx 2.718281828459$.

```
»» CONSTANT ( "E" ) ««
```

This function does not take any variable arguments.

Return type: Numeric

Example

In this example, the mathematical constant e is returned. The result is written to the log.

```
DATA _NULL_;
  e = CONSTANT ( "E" );
  PUT e=;
RUN;
```

This produces the following output:

```
e=2.718281828459
```

CONSTANT – EULER

Returns the Euler constant $\gamma \approx 0.577215664902$.

» `CONSTANT` » `("EULER")` »

This function does not take any variable arguments.

Return type: Numeric

Example

In this example, the Euler constant γ is returned. The result is written to the log.

```
DATA _NULL_;  
  gamma = CONSTANT( "EULER" );  
  PUT gamma=;  
RUN;
```

This produces the following output:

```
gamma=0.577215664902
```

CONSTANT – PI

Returns the mathematical constant $\pi \approx 3.141592653690$.

» `CONSTANT` » `("PI")` »

This function does not take any variable arguments.

Return type: Numeric

Example

In this example, the mathematical constant π is returned. The result is written to the log.

```
DATA _NULL_;  
  pi = CONSTANT( "PI" );  
  PUT pi=;  
RUN;
```

This produces the following output:

```
pi=3.141592653690
```

Numerical limits

Return platform-dependent numerical limits.

CONSTANT – BIG ↗	1760
Returns the largest number available on the current computer platform.	
CONSTANT – LOGBIG ↗	1761
Returns a logarithm of the largest number available on the current computer platform.	
CONSTANT – SQRTBIG ↗	1762
Returns the square root of the largest number available on the current computer platform.	
CONSTANT – MACEPS ↗	1763
Returns the smallest floating increment value available on the current computer platform.	
CONSTANT – LOGMACEPS ↗	1764
Returns a logarithm of the smallest floating increment value available on the current computer platform.	
CONSTANT – SQRTMACEPS ↗	1765
Returns the square root of the smallest floating increment value available on the current computer platform.	
CONSTANT – SMALL ↗	1765
Returns the smallest positive number available on the current computer platform.	
CONSTANT – LOGSMALL ↗	1766
Returns a logarithm of the smallest positive number available on the current computer platform.	
CONSTANT – SQRTSMALL ↗	1767
Returns the square root of the smallest positive number available on the current computer platform.	
CONSTANT – EXACTINT ↗	1768
Returns a platform-specific integer value for a given number of bytes.	

CONSTANT – BIG

Returns the largest number available on the current computer platform.

» `CONSTANT` » `("BIG")` »

This function does not take any variable arguments.

Return type: Numeric

Example

In this example, the largest number available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
a = CONSTANT("BIG");  
PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
a=1.797693e308
```

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – LOGBIG

Returns a logarithm of the largest number available on the current computer platform.



Determines the largest number available on the current computer platform (see also *CONSTANT – BIG* [page 1760](#)), and returns its logarithm with the specified base. If the base is omitted, the natural logarithm is returned.

Return type: Numeric

The return value is greater than one.

base

Optional argument

Type: Numeric

The base of the logarithm.

Restriction: $base > 1$

Default: e

If the argument is out of range, a missing value is returned.

Examples

In these examples, a logarithm of the largest number available on the current computer platform is returned. The results are written to the log.

```
DATA _NULL_;  
g1 = CONSTANT("LOGBIG", 1.0000003);  
PUT g1=;  
g2 = CONSTANT("LOGBIG");  
PUT g2=;  
g3 = CONSTANT("LOGBIG", CONSTANT("BIG"));  
PUT g3=;  
m1 = CONSTANT("LOGBIG", -1);  
PUT m1=;  
m2 = CONSTANT("LOGBIG", 1);  
PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=2365942731.6  
g2=709.78271289  
g3=1  
m1=.  
m2=.
```

The return value in *g3* is rounded to the asymptotic limit.

The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – SQRTBIG

Returns the square root of the largest number available on the current computer platform.

» **CONSTANT** » ("SQRTBIG") »

This function does not take any variable arguments.

See also *CONSTANT – BIG* [↗](#) (page 1760).

Return type: Numeric

Example

In this example, the square root of the largest number available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
  a = CONSTANT("SQRTBIG");  
  PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
a=1.340781e154
```

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – MACEPS

Returns the smallest floating increment value available on the current computer platform.

→ **CONSTANT** → ("MACEPS") →

This function does not take any variable arguments.

The *smallest floating increment value* is the difference between 1.0 and the next floating point number representable on the current computer platform. This increment is also known as the *machine epsilon*.

Return type: Numeric

Example

In this example, the smallest floating increment value available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
  a = CONSTANT("MACEPS");  
  PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

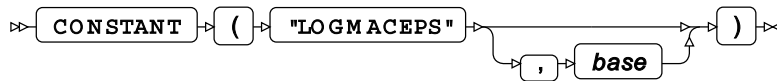
```
a=2.220446e-16
```

Note:

Other platforms might have different machine epsilon values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – LOGMACEPS

Returns a logarithm of the smallest floating increment value available on the current computer platform.



Determines the smallest floating increment value available on the current computer platform and returns its logarithm with the specified base. If the base is omitted, the natural logarithm is returned.

The *smallest floating increment value* is the difference between 1.0 and the next floating point number representable on the current computer platform. This increment is also known as the *machine epsilon*.

See also [CONSTANT – MACEPS](#) (page 1763).

Return type: Numeric

The return value is negative.

base

Optional argument

Type: Numeric

The base of the logarithm.

Restriction: *base* > 1

Default: *e*

If the argument is out of range, a missing value is returned.

Examples

In these examples, a logarithm of the smallest floating increment value available on the current computer platform is returned. The results are written to the log.

```

DATA _NULL_;
g1 = CONSTANT("LOGMACEPS", 1.0000003);
PUT g1;
g2 = CONSTANT("LOGMACEPS");
PUT g2;
m1 = CONSTANT("LOGMACEPS", -1);
PUT m1;
m2 = CONSTANT("LOGMACEPS", 1);
PUT m2;
RUN;

```

On a Windows 64-bit computer, this produces the following output:

```

g1=-120145529.3
g2=-36.04365339
m1=.
m2=.

```

The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different machine epsilon values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – SQRTMACEPS

Returns the square root of the smallest floating increment value available on the current computer platform.

➤ **CONSTANT** ➤ ("SQRTMACEPS") ➤

This function does not take any variable arguments.

Determines the smallest floating increment value available on the current computer platform and returns its square root. The *smallest floating increment value* is the difference between 1.0 and the next floating point number representable on the current computer platform. This increment is also known as the *machine epsilon*.

See also *CONSTANT – MACEPS* [↗](#) (page 1763).

Return type: Numeric

Example

In this example, the square root of the smallest floating increment value available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
  a = CONSTANT( "SQRTMACEPS" );  
  PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
a=1.4901161e-8
```

Note:

Other platforms might have different machine epsilon values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – SMALL

Returns the smallest positive number available on the current computer platform.

➤ **CONSTANT** ➤ ("SMALL") ➤

This function does not take any variable arguments.

Return type: Numeric

Example

In this example, the smallest positive number available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
a = CONSTANT ( "SMALL" ) ;  
PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

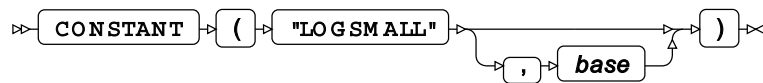
```
a=2.22507e-308
```

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – LOGSMALL

Returns a logarithm of the smallest positive number available on the current computer platform.



Determines the smallest positive number available on the current computer platform (see also *CONSTANT – SMALL* [↗](#) (page 1765)), and returns its logarithm with the specified base. If the base is omitted, the natural logarithm is returned.

Return type: Numeric

The return value is negative.

base

Optional argument

Type: Numeric

The base of the logarithm.

Restriction: $base > 1$

Default: e

If the argument is out of range, a missing value is returned.

Examples

In these examples, a logarithm of the smallest positive number available on the current computer platform is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = CONSTANT("LOGSMALL", 1.0000003);  
  PUT g1=;  
  g2 = CONSTANT("LOGSMALL");  
  PUT g2=;  
  m1 = CONSTANT("LOGSMALL", -1);  
  PUT m1=;  
  m2 = CONSTANT("LOGSMALL", 1);  
  PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=-2361321750  
g2=-708.3964185  
m1=.  
m2=.
```

The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – SQRTSMALL

Returns the square root of the smallest positive number available on the current computer platform.

» **CONSTANT** » ("SQRTSMALL") »

This function does not take any variable arguments.

See also *CONSTANT – SMALL* [↗](#) (page 1765).

Return type: Numeric

Example

In this example, the square root of the smallest positive number available on the current computer platform is returned. The result is written to the log.

```
DATA _NULL_;  
  a = CONSTANT("SQRTSMALL");  
  PUT a=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

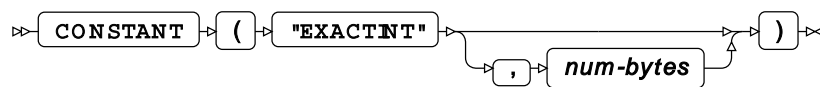
```
a=1.49167e-154
```

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

CONSTANT – EXACTINT

Returns a platform-specific integer value for a given number of bytes.



The value of *num-bytes* is rounded down to the nearest integer which must fall between 2 and 8. The resulting integer is then used for computation.

Return type: Numeric

num-bytes

Optional argument

Type: Numeric

The length of the integer in bytes.

Restriction: $2 \leq \text{num-bytes} \leq 8$

Default: 8

If the argument is out of range, a missing value is returned.

Examples

In these examples, a platform-specific integer value for a given number of bytes is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = CONSTANT("EXACTINT", 4);  
  PUT s1=;  
  s2 = CONSTANT("EXACTINT", 0);  
  PUT s2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=2097152  
s2=.
```

The last example returns a missing value because the argument is out of range.

Note:
Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

Arithmetic functions

Perform arithmetic operations on the data.

ABS ↗	1769
Returns the absolute value of a numeric argument.	
SIGN ↗	1770
Returns the sign function of a numeric argument.	
MODZ ↗	1771
Returns the modulo of two numeric arguments.	
MOD ↗	1772
Returns the modulo of two numeric arguments corrected for rounding errors.	
DIVIDE ↗	1773
Returns the ratio of two numeric arguments.	
GCD ↗	1775
Returns the greatest common divisor for a list of numeric values.	
LCM ↗	1776
Returns the least common multiple of a list of numeric values.	

ABS

Returns the absolute value of a numeric argument.



Return type: Numeric

x

- Type:** Numeric
- The point at which to calculate the absolute value.
- If the argument is out of range, a missing value is returned.

Example

In this example, the absolute value of the argument is returned. The result is written to the log.

```
DATA _NULL_;  
  a1 = ABS(-2);  
  PUT a1=;  
RUN;
```

This produces the following output:

```
a1=2
```

SIGN

Returns the sign function of a numeric argument.



Returns one of the following:

- If the argument is zero, zero is returned.
- If the argument is positive, 1 is returned.
- If the argument is negative, -1 is returned.

Return type: Numeric

x

Type: Numeric

The point at which to calculate the sign function.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the sign function of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = SIGN(0);  
  PUT s1=;  
  s2 = SIGN(-2);  
  PUT s2=;  
RUN;
```

This produces the following output:

```
s1=0  
s2=-1
```


MODZ

Returns the modulo of two numeric arguments.

⇒ **MODZ** (*dividend* , *divisor*) ⇐

Returns the remainder after division of the first argument (the *dividend*) by the second (the *divisor*). The sign of the result is the same as the sign of the dividend.

Return type: Numeric

dividend

Type: Numeric

The dividend of the fraction.

If the argument contains a missing value, a missing value is returned.

divisor

Type: Numeric

The divisor of the fraction.

Restriction: *divisor* ≠ 0

If the argument is out of range, a missing value is returned.

Examples

In these examples, the modulo of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
  m1 = MODZ( 4 , 2.2);  
  PUT m1=;  
  m2 = MODZ(-4 , 2.2);  
  PUT m2=;  
  m3 = MODZ( 4 ,-2.2);  
  PUT m3=;  
RUN;
```

This produces the following output:

```
m1=1.8  
m2=-1.8  
m3=1.8
```

The result takes on the sign of the dividend regardless of the sign of the divisor.

MOD

Returns the modulo of two numeric arguments corrected for rounding errors.

⇒ MOD ⇒ (dividend , divisor) ⇐

Returns the remainder after division of the first argument (the *dividend*) by the second (the *divisor*). The sign of the result is the same as the sign of the dividend.

This basic computation is the same as in `MODZ` function (see [MODZ](#) (page 1771)), but the result is checked for rounding errors and corrected as follows:

- If the result is nearing zero or the divisor by less than 10^{-12} , zero is returned.
- If the fractional part of the result is nearing 1 by less than 10^{-12} , the result is rounded up to the nearest integer value.
- If the fractional part of the result is nearing 0 by less than 10^{-12} , the result is rounded down to the nearest integer value.

Return type: Numeric

dividend

Type: Numeric

The dividend of the fraction.

If the argument contains a missing value, a missing value is returned.

divisor

Type: Numeric

The divisor of the fraction.

Restriction: *divisor* $\neq 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the uncorrected and corrected modulo of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
mz = MODZ(4.000000000000001,2);  
PUT mz=;  
m = MOD (4.000000000000001,2);  
PUT m=;  
RUN;
```

This produces the following output:

```
mz=9.769963e-15
m=0
```

Function `MODZ` returns an uncorrected modulo result that is very close to zero. Function `MOD` corrects it and returns zero instead.

DIVIDE

Returns the ratio of two numeric arguments.

➤ `DIVIDE` ➤ `(dividend , divisor)` ➤

Returns the result of division of the first argument (the *dividend*) by the second (the *divisor*).

Return type: Numeric

dividend

Type: Numeric

The dividend of the fraction.

divisor

Type: Numeric

The divisor of the fraction.

Restriction: *divisor* $\neq 0$

If the divisor is zero, a missing value is returned.

This function implements platform-dependent overflow and underflow tests before attempting computation:

- If the dividend underflows or the divisor overflows, zero is returned.
- If the divisor underflows, positive or negative infinity is returned where the sign is determined according to the standard rules of arithmetic.

Missing values and division by zero are processed as shown in the table below. The following missing value notations are recognised by the `DIVIDE` function:

`._` : blank

`.I` : positive infinity

`.M` : negative infinity

All other missing value notations are treated as a default missing value.

Dividend	Divisor	Result
. _	any	. _
non-missing non-zero	. I or . M	0
. I or . M	. I or . M	.
. I	negative	. M
. I	non-negative	. I
. M	negative	. I
. M	non-negative	. M
0	0	.
0	non-zero	0
positive	0	. I
negative	0	. M
other missing	any	dividend
any	other missing	divisor

Examples

In these examples, the ratio of the arguments is returned. The results are written to the log.

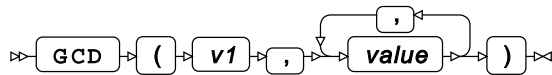
```
DATA _NULL_;
  d1 = DIVIDE( 4, 1.25 );
  PUT d1=;
  d2 = DIVIDE(-4, 0);
  PUT d2=;
  d3 = DIVIDE(.I, 2);
  PUT d3=;
  d4 = DIVIDE( 2, .I );
  PUT d4=;
  d5 = DIVIDE(.I, .M);
  PUT d5=;
  d6 = DIVIDE(.I, ._);
  PUT d6=;
  d7 = DIVIDE(.M, .Z);
  PUT d7=;
  d8 = DIVIDE(.A, 2);
  PUT d8=;
RUN;
```

This produces the following output:

```
d1=3.2
d2=.M
d3=.I
d4=0
d5=.
d6=._
d7=.Z
d8=.A
```

GCD

Returns the greatest common divisor for a list of numeric values.



Requires at least two arguments.

Return type: Numeric

The return value is a positive integer or zero.

If an overflow occurs as a result of the operation, a missing value is returned.

v1

Type: Numeric

The first value in the list.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

value

Type: Numeric

Further value to be evaluated.

Restriction: must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the greatest common divisor of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = GCD(3.3, 2, 5);  
  PUT g1=;  
  g2 = GCD(-4, -8, -24);  
  PUT g2=;  
  g3 = GCD(18, 0, 2);  
  PUT g3=;  
  g4 = GCD(18, 2);  
  PUT g4=;  
RUN;
```

This produces the following output:

```
g1=.  
g2=4  
g3=2  
g4=2
```

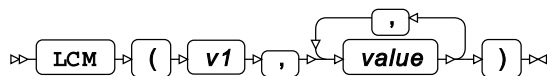
The first example returns a missing value because it has a non-integer argument.

The second example demonstrates that the GCD is always positive, even when all arguments are negative.

The third and fourth examples show that if one of the arguments is zero, it does not contribute to the calculation.

LCM

Returns the least common multiple of a list of numeric values.



Requires at least two arguments.

The least common multiple of a list of numeric values is the smallest positive integer that is divisible by each of the values in the argument list.

Return type: Numeric

The return value is positive.

If an overflow occurs as a result of the operation, a missing value is returned.

v1

Type: Numeric

The first value in the list.

Restriction: must be a non-zero integer

If the argument is out of range, a missing value is returned.

value

Type: Numeric

Further value to be evaluated.

Restriction: must be a non-zero integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the least common multiple of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = LCM(3.3, 2, 5);
  PUT g1=;
  g2 = LCM(-4, -8, -24);
  PUT g2=;
RUN;
```

This produces the following output:

```
g1=.
g2=24
```

The first example returns a missing value because it has a non-integer argument.

The second example shows that the LCM is always positive, even when all arguments are negative.

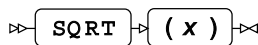
Power and exponent functions

Perform exponential operations on the data.

SQRT ↗	1777
\sqrt{x}	
Returns the square root of a numeric argument.	
EXP ↗	1778
$\exp(x) = e^x$	
Returns the exponential of a numeric argument.	
POW ↗	1779
x^y	
Returns the power function of two numeric arguments.	
CALL LOGISTIC ↗	1782
$(e^{-x} + 1)^{-1}$	
Returns the standard logistic function for each numeric variable in a list.	

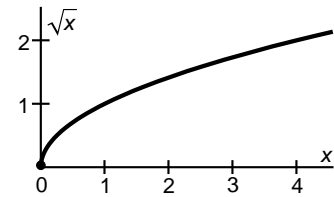
SQRT

Returns the square root of a numeric argument.



Calculates the square root of the argument:

$$\text{sqrt}(x) = \sqrt{x}$$



Return type: Numeric

The return value is positive or zero.

x

Type: Numeric

The point at which to calculate the square root.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the square root of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = SQRT(0);  
PUT s1=;  
s2 = SQRT(4);  
PUT s2=;  
RUN;
```

This produces the following output:

```
s1=0  
s2=2
```

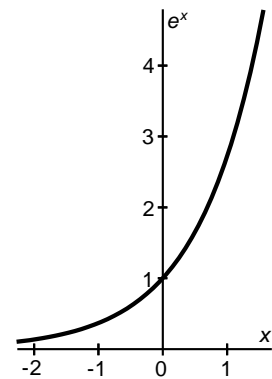
EXP

Returns the exponential of a numeric argument.

⇒ **EXP** ⇒ (x) ⇒

Raises the mathematical constant $e \approx 2.718281828459$ to the power of the argument:

$$\exp(x) = e^x$$



Return type: Numeric

The return value is positive.

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the exponential.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the exponential of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
p1 = EXP(0);  
PUT p1=;  
p2 = EXP(4);  
PUT p2=;  
RUN;
```

This produces the following output:

```
p1=1  
p2=54.598150033
```

POW

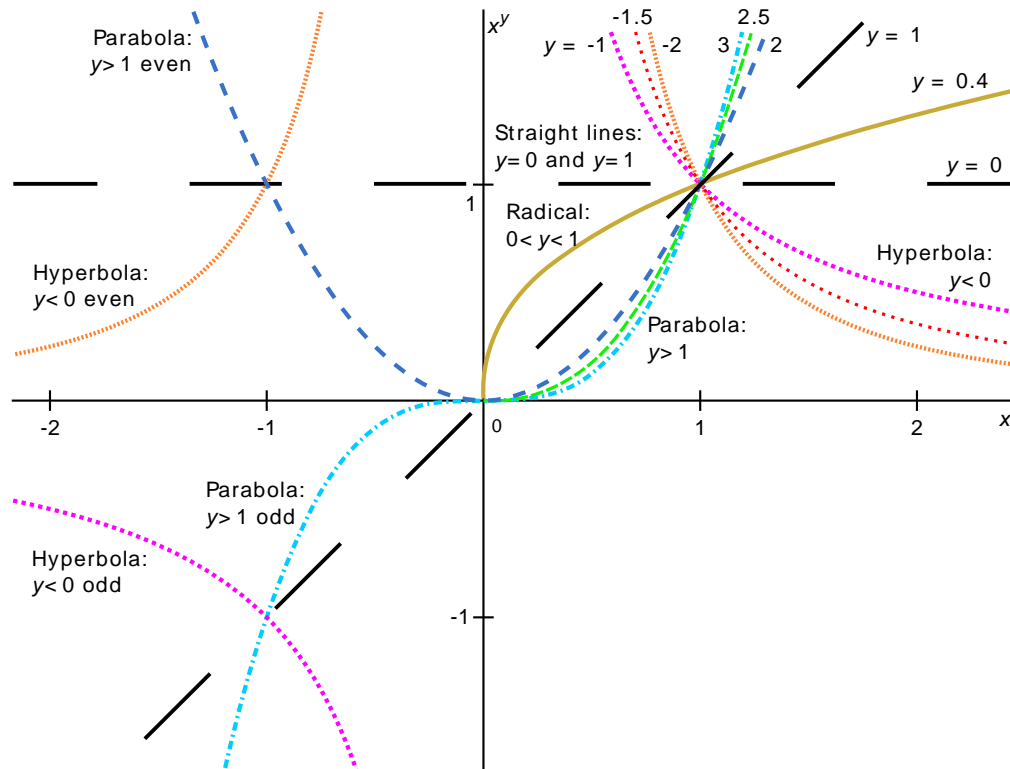
Returns the power function of two numeric arguments.

➤ **POW** ➤ (*base* , *exponent*) ➤

Raises the first argument (the *base*) to the power of the second (the *exponent*):

$$\text{pow}(x, y) = x^y$$

where x is the base and y is the exponent.



The profile of this function depends on the value of the exponent.

- For $y=0$ and $y=1$, the function is a straight line:

$$\text{pow}(x, 0) = 1$$

$$\text{pow}(x, 1) = x$$

- For $0 < y < 1$, the function is defined for $x \geq 0$ only and has the shape of a radical or a root curve.
- For $y < 0$ the curve is a hyperbola. The hyperbola is defined for all $x > 0$ and is always positive on this domain. In addition, for integer y it is also defined for $x < 0$ and is positive for even values of y and negative for odd values of y .
- For $y > 1$ the curve is a parabola. The parabola is defined for all $x \geq 0$ and is always positive on this domain. In addition, similarly to the hyperbola, for integer y it is also defined for $x < 0$ and is positive for even values of y and negative for odd values of y .

Return type: Numeric

A missing value is returned if:

- The base is negative and the exponent is not an integer.
- The base is zero and the exponent is negative.
- Any argument contains a missing value.
- An overflow occurs as a result of the operation.

On some platforms a missing value is returned if both the base and the exponent are zero.

base**Type:** Numeric

The base of the power function.

Restrictions:

- $base > 0$ if $exponent$ is not an integer
- $base \neq 0$ if $exponent < 0$

If the argument is out of range, a missing value is returned.

exponent**Type:** Numeric

The exponent of the power function.

If the argument contains a missing value, a missing value is returned.

Examples

The following examples illustrate hyperbolic and parabolic profiles of this function. The results are written to the log.

```
DATA _NULL_;
  a1 = POW(4,2);
  PUT a1=;

  h1 = POW( 2.9,-1 );
  PUT h1=;
  h2 = POW( 2.9,-1.5);
  PUT h2=;
  h3 = POW( 2.9,-2 );
  PUT h3=;
  h4 = POW(-2.9,-1 );
  PUT h4=;
  h5 = POW(-2.9,-1.5);
  PUT h5=;
  h6 = POW(-2.9,-2 );
  PUT h6=;

  p1 = POW( 2.9, 2 );
  PUT p1=;
  p2 = POW( 2.9, 2.5);
  PUT p2=;
  p3 = POW( 2.9, 3 );
  PUT p3=;
  p4 = POW(-2.9, 2 );
  PUT p4=;
  p5 = POW(-2.9, 2.5);
  PUT p5=;
  p6 = POW(-2.9, 3 );
  PUT p6=;
RUN;
```

This produces the following output:

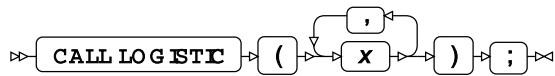
```
a1=16
h1=0.3448275862
h2=0.2024897309
h3=0.1189060642
h4=-0.344827586
h5=.
h6=0.1189060642

p1=8.41
p2=14.321713934
p3=24.389
p4=8.41
p5=.
p6=-24.389
```

The fifth example in both series returns a missing value because the function is not defined for negative bases and fractional exponents.

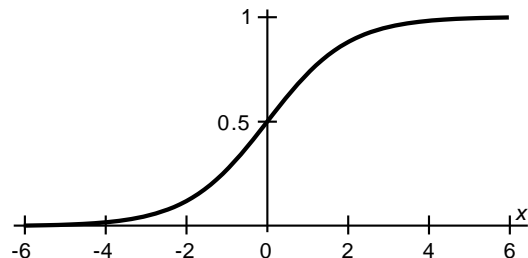
CALL LOGISTIC

Returns the standard logistic function for each numeric variable in a list.



Calculates the standard logistic function for each variable in the argument list replacing the input values in the variables with the result of the calculation.

$$f(x) = (e^{-x} + 1)^{-1}$$



The return values are between 0 and 1, not including the bounds.

If any argument contains a missing value, all values are set to missing.

x

Type: Numeric

Examples

In this example, the standard logistic function for each variable in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  s1=0; s2=1.1; s3="4"; s4=-6.6;
  CALL LOGISTIC (s1,s2,s3,s4);

  PUT s1=; PUT s2=; PUT s3=; PUT s4=;

  m1=0; m2=1.1; m3="four"; m4=-6.6;
  CALL LOGISTIC (m1,m2,m3,m4);

  PUT m1=; PUT m2=; PUT m3=; PUT m4=;
RUN;
```

This produces the following output:

```
s1=0.5
s2=0.7502601056
s3=0.98201379
s4=0.00135852

m1=.
m2=.
m3=.
m4=.
```

The argument value "4" has been converted into a number, but the argument value "four" is considered missing. Therefore all values are set to missing in the second example series.

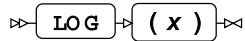
Logarithmic functions

Perform logarithmic operations on the data.

LOG ↗	1784
log (x)	
Returns the natural logarithm of a numeric argument.	
LOG2 ↗	1785
log ₂ (x)	
Returns the base 2 logarithm of a numeric argument.	
LOG10 ↗	1786
log ₁₀ (x)	
Returns the base 10 logarithm of a numeric argument.	
LOG1PX ↗	1787
log (x+1)	
Returns the natural logarithm of a numeric argument augmented by one.	

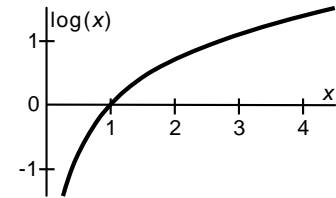
LOG

Returns the natural logarithm of a numeric argument.



Calculates the natural logarithm of the argument:

$$\log(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
g1 = LOG(0.003);  
PUT g1=;  
g2 = LOG(1);  
PUT g2=;  
g3 = LOG(2.3);  
PUT g3=;  
g4 = LOG(1e-307);  
PUT g4=;  
m1 = LOG(0);  
PUT m1=;  
m2 = LOG(-2.2);  
PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=-5.80914299  
g2=0  
g3=0.8329091229  
g4=-706.8936235  
m1=.  
m2=.
```

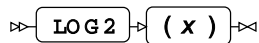
The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

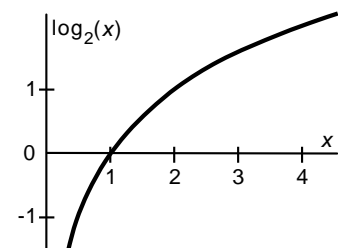
LOG2

Returns the base 2 logarithm of a numeric argument.



Calculates the base 2 logarithm of the argument:

$$\log_2(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the base 2 logarithm of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = LOG2(0.003);  
  PUT g1=;  
  g2 = LOG2(1);  
  PUT g2=;  
  g3 = LOG2(2.3);  
  PUT g3=;  
  g4 = LOG2(1e-307);  
  PUT g4=;  
  m1 = LOG2(0);  
  PUT m1=;  
  m2 = LOG2(-2.2);  
  PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=-8.380821784
g2=0
g3=1.2016338612
g4=-1019.831925
m1=.
m2=.
```

The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

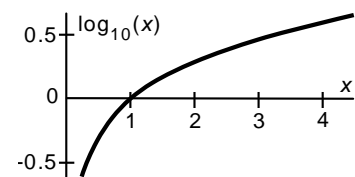
LOG10

Returns the base 10 logarithm of a numeric argument.

» LOG10 » (x) »

Calculates the base 10 logarithm of the argument:

$$\log_{10}(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the natural logarithm.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the base 10 logarithm of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = LOG10(0.003);  
  PUT g1=;  
  g2 = LOG10(1);  
  PUT g2=;  
  g3 = LOG10(2.3);  
  PUT g3=;  
  g4 = LOG10(1e-307);  
  PUT g4=;  
  m1 = LOG10(0);  
  PUT m1=;  
  m2 = LOG10(-2.2);  
  PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=-2.522878745  
g2=0  
g3=0.361727836  
g4=-307  
m1=.  
m2=.
```

The last two examples return a missing value because the argument is out of range.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

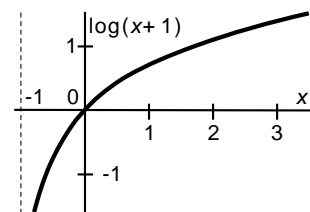
LOG1PX

Returns the natural logarithm of a numeric argument augmented by one.

» LOG1PX » (x) »

Calculates the natural logarithm of a numeric argument augmented by one:

$$\log(x+1)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the calculation.

Restriction: $x > -1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of a numeric argument augmented by one is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = LOG1PX(0.003);
  PUT g1=;
  g2 = LOG1PX(0);
  PUT g2=;
  g3 = LOG1PX(-0.3);
  PUT g3=;
  m1 = LOG1PX(-1);
  PUT m1=;
  m2 = LOG1PX(-2.2);
  PUT m2=;
RUN;
```

This produces the following output:

```
g1=0.002995509
g2=0
g3=-0.356674944
m1=
m2=
```

The last two examples return a missing value because the argument is out of range.

Trigonometric functions

Perform trigonometric operations on the data.

Many of the functions in this section have special values or poles at multiples or fractions of π , where $\pi \approx 3.141592653690$, both positive and negative. However, as π is an irrational number, it cannot be represented exactly in a computer application. Therefore intervals with fractions of π as bounds are always open, while poles or asymptotes are never reached.

Primary trigonometric functions ↗	1789
Perform primary trigonometric operations on the data.	
Inverse trigonometric functions ↗	1793
Perform inverse trigonometric operations on the data.	

Hyperbolic functions 1800
 Perform hyperbolic operations on the data.

Inverse hyperbolic functions 1805
 Perform inverse hyperbolic operations on the data.

Primary trigonometric functions

Perform primary trigonometric operations on the data.

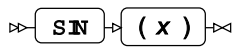
SIN 1789
 sin (x)
 Returns the sine of a numeric argument.

COS 1790
 cos (x)
 Returns the cosine of a numeric argument.

TAN 1792
 tan (x)
 Returns the tangent of a numeric argument.

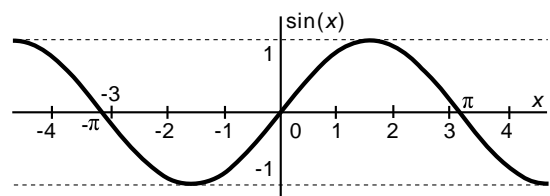
SIN

Returns the sine of a numeric argument.



Calculates the sine of the argument:

sin (x)



Return type: Numeric

The return value is between -1 and 1, inclusive.

x

Type: Numeric

The point at which to calculate the sine.

If the argument contains a missing value, a missing value is returned.

Basic examples

In these examples, the sine of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = SIN(-0.5);  
PUT s1=;  
s2 = SIN(1.1);  
PUT s2=;  
s3 = SIN(0);  
PUT s3=;  
RUN;
```

This produces the following output:

```
s1=-0.479425539  
s2=0.8912073601  
s3=0
```

Examples — using values close to function extremums

In these examples, the sine is applied to the input values close to the function extremums: π and $\pi/2$. The results are written to the log.

```
DATA _NULL_;  
p1 = SIN(3.1415926535897);  
PUT p1=;  
p2 = SIN(1.5707963267949);  
PUT p2=;  
p3 = SIN(1.5707963267948);  
PUT p3=;  
RUN;
```

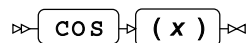
This produces the following output:

```
p1=9.33812e-14  
p2=1  
p3=1
```

The first example uses a value close to π as input and returns a value close to 0. However, the second and third examples use similar approximations of $\pi/2$, but return exactly 1 in each case. This behaviour is platform-specific and may depend on the degree of approximation of the input.

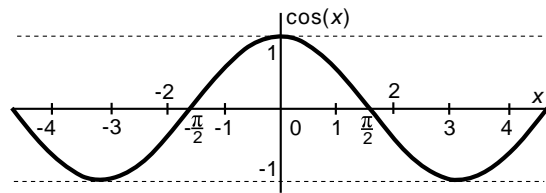
COS

Returns the cosine of a numeric argument.



Calculates the cosine of the argument:

$\cos(x)$



Return type: Numeric

The return value is between -1 and 1, inclusive.

x

Type: Numeric

The point at which to calculate the cosine.

If the argument contains a missing value, a missing value is returned.

Basic examples

In these examples, the cosine of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = COS(-0.5);  
  PUT s1=;  
  s2 = COS(1.1);  
  PUT s2=;  
  s3 = COS(0);  
  PUT s3=;  
RUN;
```

This produces the following output:

```
s1=0.8775825619  
s2=0.4535961214  
s3=1
```

Examples — using values close to function extremums

In these examples, the cosine is applied to the input values close to the function extremums: π and $\pi/2$.

The results are written to the log.

```
DATA _NULL_;  
  p1 = COS(3.1415926535897);  
  PUT p1=;  
  p2 = COS(1.5707963267949);  
  PUT p2=;  
  p3 = COS(1.5707963267948);  
  PUT p3=;  
RUN;
```

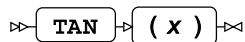
This produces the following output:

```
p1=-1  
p2=-3.49148e-15  
p3=9.665064e-14
```

The first example uses a value close to π as input and returns exactly -1. However, the second and third examples use similar approximations of $\pi/2$, but return approximations of 0, with the correct sign. This behaviour is platform-specific and may depend on the degree of approximation of the input.

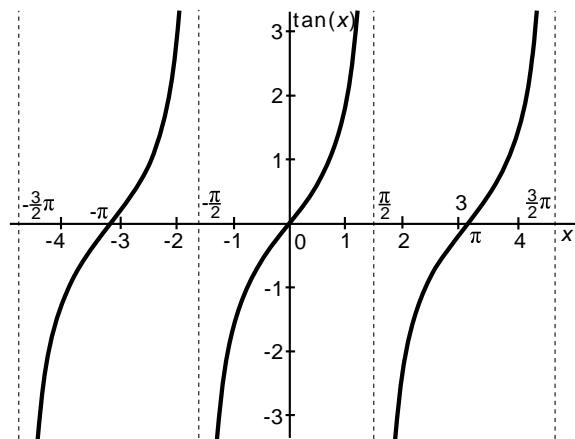
TAN

Returns the tangent of a numeric argument.



Calculates the tangent of the argument:

$$\tan(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the tangent.

If the argument contains a missing value, a missing value is returned.

The mathematical tangent function has poles at multiples of $\pi/2$, where $\pi \approx 3.141592653690$, both positive and negative. However, as π is an irrational number, it cannot be represented exactly in a computer application. Therefore, function `TAN` never hits a pole and remains defined for all numeric values.

Basic examples

In this example, the tangent of the argument is returned. The result is written to the log.

```
DATA _NULL_;
  s1 = TAN(-0.5);
  PUT s1=;
  s2 = TAN(1.1);
  PUT s2=;
  s3 = TAN(0);
  PUT s3=;
RUN;
```

This produces the following output:

```
s1=-0.54630249
s2=1.9647596572
s3=0
```

Examples — using values close to function extremums

In these examples, the tangent is applied to the input values close to the function extremums: π and $\pi/2$. The results are written to the log.

```
DATA _NULL_;
  p1 = TAN(3.1415926535897);
  PUT p1=;
  p2 = TAN(1.5707963267949);
  PUT p2=;
  p3 = TAN(1.5707963267948);
  PUT p3=;
RUN;
```

This produces the following output:

```
p1=-9.33812e-14
p2=-2.8641138e14
p3=1.034654e13
```

The first example uses a value close to π as input and returns a value close to 0. The second and third examples use similar approximations of $\pi/2$, and return very large values with the correct sign approximating positive and negative infinity.

Inverse trigonometric functions

Perform inverse trigonometric operations on the data.

ARSIN ↗	1794
$\alpha = \arcsin(x)$	
Returns the principal value of inverse sine of a numeric argument.	
ARCOS ↗	1795
$\alpha = \arccos(x)$	

Returns the principal value of inverse cosine of a numeric argument.

ATAN [↗](#) 1796

$$\alpha = \arctan(x)$$

Returns the principal value of arc tangent of a numeric argument.

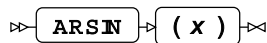
ATAN2 [↗](#) 1797

$$\alpha = \arctan\left(\frac{y}{x}\right)$$

Returns the signed angle between the positive x-axis and a vector represented by two numeric arguments.

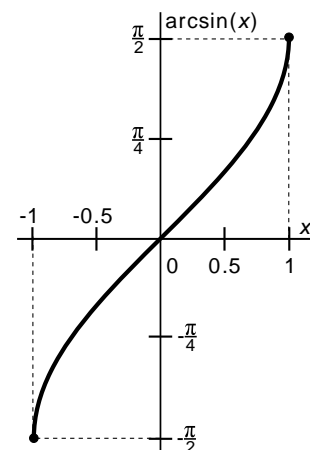
ARSIN

Returns the principal value of inverse sine of a numeric argument.



Calculates the principal value of inverse sine of the argument in radians, where the inverse sine is defined as a trigonometrical inverse of sine:

$$\alpha = \arcsin(x) \text{ is defined such that } x = \sin(\alpha)$$



Return type: Numeric

The return value is between $-\pi/2$ and $\pi/2$.

x

Type: Numeric

The point at which to calculate the principal value of inverse sine.

Restriction: $-1 \leq x \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the principal value of inverse sine of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = ARSIN(-1);
  PUT s1=;
  s2 = ARSIN(0);
  PUT s2=;
  s3 = ARSIN(1);
  PUT s3=;
  s4 = ARSIN(1.5);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=-1.570796327
s2=0
s3=1.570796327
s4=.
```

The last example returns a missing value because the argument is out of range.

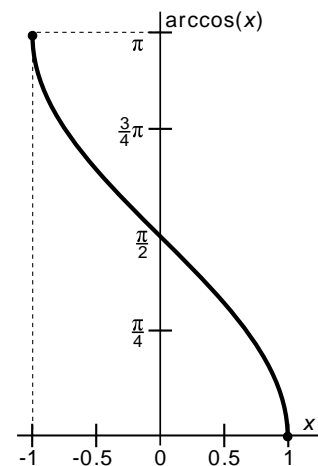
ARCOS

Returns the principal value of inverse cosine of a numeric argument.

⇒ **ARCOS** ⇒ (*x*) ⇒

Calculates the principal value of inverse cosine of the argument in radians, where the inverse cosine is defined as a trigonometrical inverse of cosine:

$$\alpha = \arccos(x) \text{ is defined such that } x = \cos(\alpha)$$



Return type: Numeric

The return value is between 0 and π , inclusive.

x

Type: Numeric

The point at which to calculate the principal value of inverse sine.

Restriction: $-1 \leq x \leq 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the principal value of inverse cosine of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = ARCOS(-1);
  PUT s1=;
  s2 = ARCOS(0);
  PUT s2=;
  s3 = ARCOS(1);
  PUT s3=;
  s4 = ARCOS(1.5);
  PUT s4=;
RUN;
```

This produces the following output:

```
s1=3.1415926536
s2=1.5707963268
s3=0
s4=.
```

The last example returns a missing value because the argument is out of range.

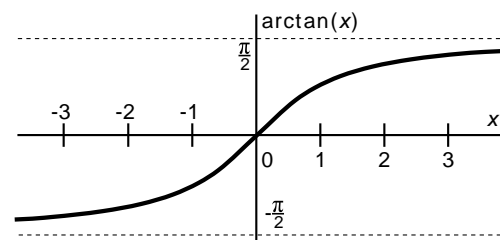
ATAN

Returns the principal value of arc tangent of a numeric argument.

➤ **ATAN** ➤ (**x**) ➤

Calculates the principal value of arc tangent of the argument in radians, where the arc tangent is defined as a trigonometrical inverse of tangent:

$\alpha = \arctan(x)$ is defined such that $x = \tan(\alpha)$



Return type: Numeric

The return value is between $-\pi/2$ and $\pi/2$.

x

Type: Numeric

The point at which to calculate the principal value of arc tangent.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the principal value of arc tangent of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
s1 = ATAN(0);  
PUT s1=;  
s2 = ATAN(1e308);  
PUT s2=;  
s3 = ATAN(-1e308);  
PUT s3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=0  
s2=1.5707963268  
s3=-1.5707963268
```

For very large values, positive and negative, as shown in examples 2 and 3, function `ATAN` returns an approximation of $\pi/2$ with the appropriate sign.

Note:

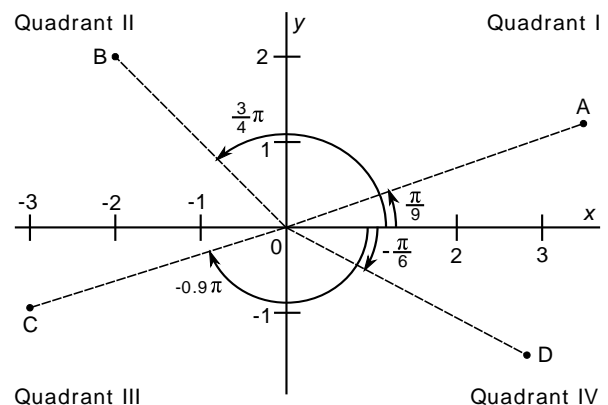
Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

ATAN2

Returns the signed angle between the positive x -axis and a vector represented by two numeric arguments.

» `ATAN2` » `(y , x)` »

Calculates the signed angle between the positive x -axis and a vector (x,y) in the range between $-\pi$ and π . The sign of the angle returned represents the quadrant of the end point of the vector.



For return values between $-\pi/2$ and $\pi/2$, this calculation is equivalent to the arc tangent of the ratio of the arguments, see [ATAN](#) (page 1796):

$$\alpha = \arctan\left(\frac{y}{x}\right)$$

A = (3.35, 1.22)
C = (-2.953, -0.96)

B = (-2, 2)
D = (2.77, -1.6)

Attention:

The order of the function arguments is the inverse of the order of coordinates of the vector they represent. The argument order is y, x.

Return type: Numeric

The return value is between $-\pi$ and π .

If both arguments are zero, zero is returned.

y

Type: Numeric

The Y coordinate value.

If the argument contains a missing value, a missing value is returned.

x

Type: Numeric

The X coordinate value.

If the argument contains a missing value, a missing value is returned.

Function `ATAN2` has the following advantages compared to the classic arc tangent function `ATAN`:

- `ATAN2` correctly returns $\pi/2$ or $-\pi/2$ for points on the y-axis where `ATAN` is undefined.
- `ATAN2` differentiates between diametrically opposite quadrants, which is not possible for `ATAN`. Thus, `ATAN2` returns distinctly signed values for each quadrant where `ATAN` cannot distinguish between points in quadrants I and III, and quadrants II and IV.

Basic examples

In these examples, the signed angle between the positive x-axis and a vector of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
  a  = ATAN2( 1.22, 3.35 );  
  PUT a=;  
  b  = ATAN2( 2,    -2    );  
  PUT b=;  
  c  = ATAN2(-0.96,-2.953);  
  PUT c=;  
  d  = ATAN2(-1.6,  2.77 );  
  PUT d=;  
RUN;
```

This produces the following output:

```
a=0.3492502751  
b=2.3561944902  
c=-2.827276528  
d=-0.523799047
```

These results can be approximated as follows: $a \approx \pi/9$, $b \approx 3/4\pi$, $c \approx -0.9\pi$, $d \approx -\pi/6$.

Examples – points where the function changes its sign

The following examples demonstrate calculations for points on the coordinate axes, as well as points near the x-axis where the return value changes its sign. The results are written to the log.

```
DATA _NULL_;  
  s1 = ATAN2( 0, 0 );  
  PUT s1=;  
  s2 = ATAN2( 1, 0 );  
  PUT s2=;  
  s3 = ATAN2(-1, 0 );  
  PUT s3=;  
  s4 = ATAN2( 0, 1 );  
  PUT s4=;  
  s5 = ATAN2( 0,-1 );  
  PUT s5=;  
  s6 = ATAN2( 1e-307, 1 );  
  PUT s6=;  
  s7 = ATAN2(-1e-307, 1 );  
  PUT s7=;  
  s8 = ATAN2( 1e-307,-1 );  
  PUT s8=;  
  s9 = ATAN2(-1e-307,-1 );  
  PUT s9=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=0
s2=1.5707963268
s3=-1.5707963268
s4=0
s5=3.1415926536
s6=1e-307
s7=-1e-307
s8=3.1415926536
s9=-3.1415926536
```

Note:
Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

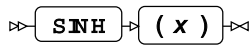
Hyperbolic functions

Perform hyperbolic operations on the data.

SINH ↗	1800
$\frac{1}{2}(e^x - e^{-x})$	
Returns the hyperbolic sine of a numeric argument.	
COSH ↗	1802
$\frac{1}{2}(e^x + e^{-x})$	
Returns the hyperbolic cosine of a numeric argument.	
TANH ↗	1803
$\frac{\sinh(x)}{\cosh(x)}$	
Returns the hyperbolic tangent of a numeric argument.	
CALL TANH ↗	1804
$\frac{\sinh(x)}{\cosh(x)}$	
Calculates the hyperbolic tangent for each numeric variable in a list.	

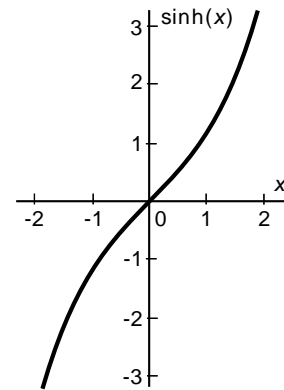
SINH

Returns the hyperbolic sine of a numeric argument.



Calculates the hyperbolic sine of the argument:

$$\sinh(x) = \frac{1}{2}(e^x - e^{-x})$$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the hyperbolic sine.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the hyperbolic sine of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = SINH(-1.7);  
  PUT s1=;  
  s2 = SINH(0);  
  PUT s2=;  
  s3 = SINH(100);  
  PUT s3=;  
  s4 = SINH(800);  
  PUT s4=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=-2.645631934  
s2=0  
s3=1.3440586e43  
s4=.
```

The last example returns a missing value because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

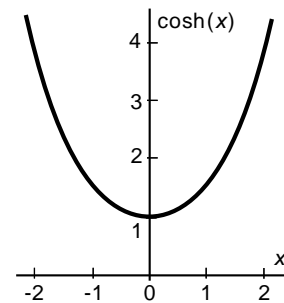
COSH

Returns the hyperbolic cosine of a numeric argument.

➤ **COSH** ➤ **(x)** ➤

Calculates the hyperbolic cosine of the argument:

$$\cosh(x) = \frac{1}{2}(e^x + e^{-x})$$



Return type: Numeric

The return value is greater than or equal to one.

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the hyperbolic cosine.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the hyperbolic cosine of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = COSH(-1.7);  
  PUT s1=;  
  s2 = COSH(0);  
  PUT s2=;  
  s3 = COSH(100);  
  PUT s3=;  
  s4 = COSH(800);  
  PUT s4=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=2.8283154579  
s2=1  
s3=1.3440586e43  
s4=.
```

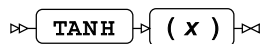
The last example returns a missing value because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

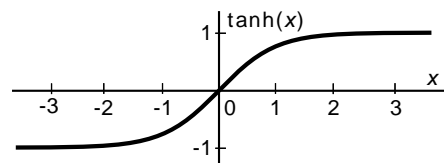
TANH

Returns the hyperbolic tangent of a numeric argument.



Calculates the hyperbolic tangent of the argument:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$



Return type: Numeric

The return value is between -1 and 1, not including the bounds.

x

Type: Numeric

The point at which to calculate the hyperbolic tangent.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the hyperbolic tangent of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = TANH(-1.7);  
  PUT s1=;  
  s2 = TANH(0);  
  PUT s2=;  
  s3 = TANH(15);  
  PUT s3=;  
RUN;
```

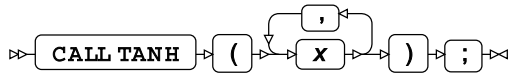
This produces the following output:

```
s1=-0.935409071  
s2=0  
s3=1
```

The return value in the last example is rounded to the asymptotic limit.

CALL TANH

Calculates the hyperbolic tangent for each numeric variable in a list.



Calculates the hyperbolic tangent for each variable in the argument list replacing the input values in the variables with the result of the calculation.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

See [TANH](#) (page 1803) for details on the hyperbolic tangent.

The return values are between -1 and 1, not including the bounds.

If any argument contains a missing value, all values are set to missing.

x

Type: Numeric

The point at which to calculate the hyperbolic tangent.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the hyperbolic tangent for each of the arguments is returned. The results are written to the log.

```

DATA _NULL_;
  s1=0.6; s2=1; s3="4";
  CALL TANH (s1,s2,s3);

  PUT s1=;

  PUT s2=;

  PUT s3=;

  m1=0.6; m2=1; m3="four";
  CALL TANH (m1,m2,m3);

  PUT m1=;

  PUT m2=;

  PUT m3=;
RUN;

```

This produces the following output:

```
s1=0.537049567
s2=0.761594156
s3=0.9993292997

m1=.
m2=.
m3=.
```

The argument value "4" has been converted into a number, but the argument value "four" is considered missing. Therefore all values are set to missing in the second example series.

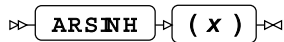
Inverse hyperbolic functions

Perform inverse hyperbolic operations on the data.

ARSINH ↗	1805
$\log \left(x + \sqrt{x^2 + 1} \right)$	
Returns the inverse hyperbolic sine of a numeric argument.	
ARCOSH ↗	1806
$\log \left(x + \sqrt{x^2 - 1} \right)$	
Returns the inverse hyperbolic cosine of a numeric argument.	
ARTANH ↗	1808
$\frac{1}{2} \log \left(\frac{1+x}{1-x} \right)$	
Returns the inverse hyperbolic tangent of a numeric argument.	

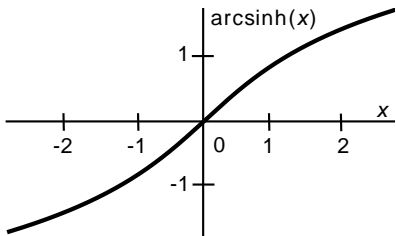
ARSINH

Returns the inverse hyperbolic sine of a numeric argument.



Calculates the inverse hyperbolic sine of the argument:

$$\operatorname{arsinh}(x) = \log \left(x + \sqrt{x^2 + 1} \right)$$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x**Type:** Numeric

The point at which to calculate the inverse hyperbolic sine.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the inverse hyperbolic sine of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = ARSINH(-1.7);
  PUT s1=;
  s2 = ARSINH(0);
  PUT s2=;
  s3 = ARSINH(100);
  PUT s3=;
  s4 = ARSINH(1e200);
  PUT s4=;
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
s1=-1.300820427
s2=0
s3=5.2983423656
s4=.
```

The last example returns a missing value because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

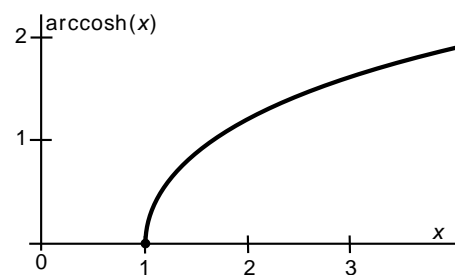
ARCOSH

Returns the inverse hyperbolic cosine of a numeric argument.

⇒ **ARCOSH** ⇒ **(x)** ⇒

Calculates the inverse hyperbolic cosine of the argument:

$$\operatorname{arcosh}(x) = \log(x + \sqrt{x^2 - 1})$$



Return type: Numeric

The return value is positive or zero.

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the inverse hyperbolic cosine.

Restriction: $x \geq 1$

If the argument contains a missing value, a missing value is returned.

Basic examples

In these examples, the inverse hyperbolic cosine of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  s1 = ARCOSH(1);  
  PUT s1=;  
  s2 = ARCOSH(1.1);  
  PUT s2=;  
RUN;
```

This produces the following output:

```
s1=0  
s2=0.4435682544
```

Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;  
  c1 = ARCOSH(-1.7);  
  PUT c1=;  
  c2 = ARCOSH(0);  
  PUT c2=;  
  c3 = ARCOSH(1e200);  
  PUT c3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
c1=.  
c2=.  
c3=.
```

The above examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

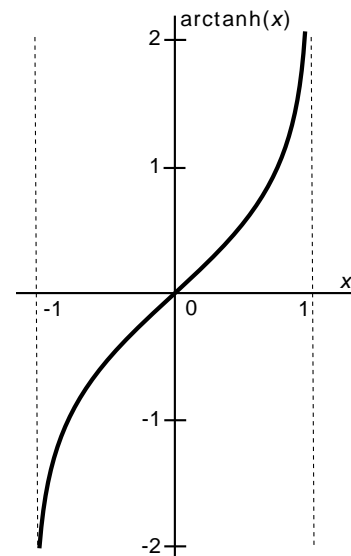
ARTANH

Returns the inverse hyperbolic tangent of a numeric argument.

➤ **ARTANH** ➤ (*x*) ➤

Calculates the inverse hyperbolic tangent of the argument:

$$\operatorname{artanh}(x) = \frac{1}{2} \log \left(\frac{1+x}{1-x} \right)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the inverse hyperbolic tangent.

Restriction: $-1 < x < 1$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the inverse hyperbolic tangent of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = ARTANH(-0.3);
  PUT s1=;
  s2 = ARTANH(0);
  PUT s2=;
  s3 = ARTANH(0.9999999999);
  PUT s3=;
  s4 = ARTANH(1);
  PUT s4=;
  s5 = ARTANH(-5.1);
  PUT s5=;
RUN;
```

This produces the following output:

```
s1=-0.309519604
s2=0
s3=11.859499014
s4=.
s5=.
```

The last two examples return a missing value because the argument is out of range.

Factorials and special functions

Perform factorial operations on the data and calculate special functions: Beta, Gamma, Bessel, Airy, Error and related functions.

The factorial function and two types of Euler integral have the following relationships to each other:

For integer values:

$$\Gamma(n) = (n-1)!$$

$$B(x, y) = \frac{(x-1)!(y-1)!}{(x+y-1)!}$$

For all values:

$$B(x, y) = \frac{\Gamma(x) \Gamma(y)}{\Gamma(x+y)}$$

$$\Gamma(x) \Gamma(y) = B(x, y) \Gamma(x+y)$$

FACT [↗](#)..... 1811
 $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$

Returns the factorial of a numeric argument.

LFACT [↗](#)..... 1812
 $\log n!$

Returns the natural logarithm of the factorial of a numeric argument.

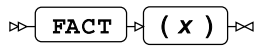
GAMMA [↗](#)..... 1813
 $\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$

Returns the Gamma function of a numeric argument, or the Euler integral of the second kind.	
LGAMMA ↗	1815
$\log \Gamma (x)$	
Returns the natural logarithm of the Gamma function of a numeric argument.	
DIGAMMA ↗	1816
$\psi (x) = \frac{d}{dx} \log \Gamma (x)$	
Returns the Digamma function of a numeric argument.	
TRIGAMMA ↗	1818
$\psi_1(x) = \frac{d^2}{dx^2} \log \Gamma (x)$	
Returns the Trigamma function of a numeric argument.	
BETA ↗	1819
$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$	
Returns the Beta function of two numeric arguments, or the Euler integral of the first kind.	
LOGBETA ↗	1821
$\log B(x, y)$	
Returns the natural logarithm of the Beta function of two numeric arguments.	
JBESSEL ↗	1823
$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$	
Returns the Bessel function of the first kind for two numeric arguments.	
IBESSEL ↗	1825
$I_\alpha(x) = \sum_{m=0}^{\infty} \frac{1}{m! \Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$	
Returns the modified Bessel function of the first kind for two numeric arguments, scaled or unscaled.	
AIRY ↗	1827
$Ai(x) = \frac{1}{\pi} \int_0^{\infty} \cos\left(\frac{t^3}{3} + xt\right) dt$	
Returns the Airy function of the first kind for a numeric argument.	
DAIRY ↗	1829
$Ai'(x)$	
Returns the first derivative of the Airy function of the first kind for a numeric argument.	
ERF ↗	1830
$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$	
Returns the error function of a numeric argument.	

ERFC ↗	1831
$\text{erfc}(x) = 1 - \text{erf}(x)$ Returns the complementary error function of a numeric argument.	

FACT

Returns the factorial of a numeric argument.



Returns the factorial of the argument, or the product of all integer numbers from 1 up to and including the value of the argument:

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

Note:
The upper limit for the argument is platform-dependent.

x

Type: Numeric

The point at which to calculate the factorial.

Restriction: $x \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Examples

In these examples, the factorial of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  f1 = FACT(0);
  PUT f1=;
  f2 = FACT(-2);
  PUT f2=;
  f3 = FACT(200);
  PUT f3=;
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
f1=1
f2=.
f3=.
```

The argument in the second example is negative, whereas the third example causes an overflow during multiplication.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

LFACT

Returns the natural logarithm of the factorial of a numeric argument.

⇒ **LFACT** (*x*) ⇐

Calculates the natural logarithm of the factorial, see *FACT* [↗](#) (page 1811):

$$\log n!$$

Function **LFACT** allows much larger numbers as input than function **FACT** due to the use of the logarithm.

Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

Note:

The upper limit for the argument is platform-dependent.

x

Type: Numeric

The point at which to calculate the factorial.

Restriction: $x \geq 0$ must be integer

If the argument is out of range, a missing value is returned.

Basic examples

In these examples, the natural logarithm of the factorial of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  f1 = LFACT(0);  
  PUT f1=;  
  f2 = LFACT(1);  
  PUT f2=;  
RUN;
```

This produces the following output:

```
f1=0  
f2=0
```

Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;  
  m1 = LFACT(-2);  
  PUT m1=;  
  m2 = LFACT(2.3);  
  PUT m2=;  
  m3 = LFACT(2.2e9);  
  PUT m3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
m1= .  
m2= .  
m3= .
```

The above examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

GAMMA

Returns the Gamma function of a numeric argument, or the Euler integral of the second kind.

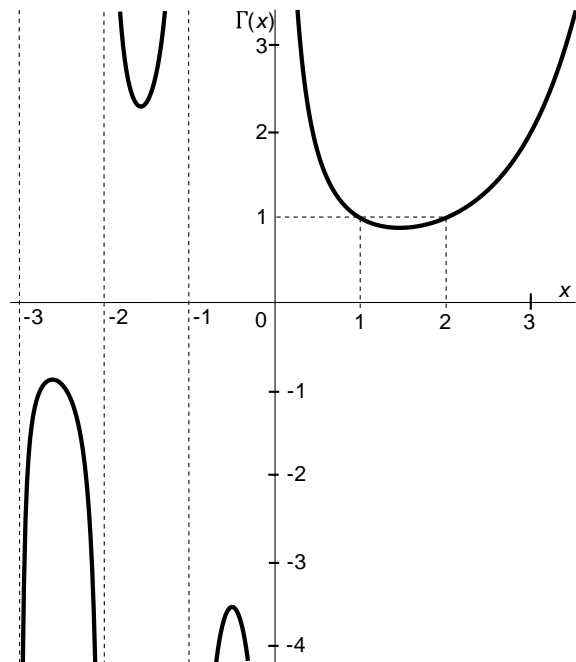
➤ **GAMMA** ➤ (*x*) ➤

The Gamma function is defined via a convergent improper integral:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

For integers, this equals the factorial (see [FACT](#) (page 1811)) of the argument reduced by one:

$$\Gamma(n) = (n-1)!$$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the Gamma function.

Restriction: cannot be within 10^{-12} of a negative integer or zero

If the argument is out of range, a missing value is returned.

Basic examples

In these examples, the Gamma function of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = GAMMA(4.2);
  PUT g1=;
  g2 = GAMMA(0.003);
  PUT g2=;
  g3 = GAMMA(-0.3);
  PUT g3=;
  g4 = GAMMA(-1.5);
  PUT g4=;
  g5 = GAMMA(-2.7);
  PUT g5=;
RUN;
```

This produces the following output:

```
g1=7.7566895358  
g2=332.7590767  
g3=-4.326851109  
g4=2.3632718012  
g5=-0.931082785
```

Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;  
  m1 = GAMMA(0);  
  PUT m1=;  
  m2 = GAMMA(1e-13);  
  PUT m2=;  
  m3 = GAMMA(-2);  
  PUT m3=;  
  m4 = GAMMA(1e306);  
  PUT m4=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
m1= .  
m2= .  
m3= .  
m4= .
```

The above examples return a missing value because the argument is nearing a function pole, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

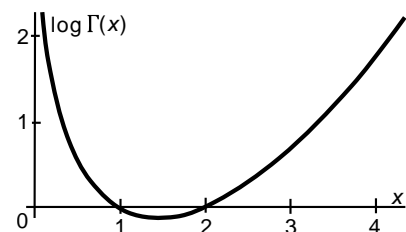
LGAMMA

Returns the natural logarithm of the Gamma function of a numeric argument.

➤ **LGAMMA** ➤ (*x*) ➤

Calculates the natural logarithm of the Gamma function, see [GAMMA](#) (page 1813):

$$\log \Gamma(x)$$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the natural logarithm of the Gamma function.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the Gamma function of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = LGAMMA(0.003);  
  PUT g1=;  
  g2 = LGAMMA(1);  
  PUT g2=;  
  g3 = LGAMMA(1.3);  
  PUT g3=;  
  g4 = LGAMMA(3.1);  
  PUT g4=;  
  
  m1 = LGAMMA(0);  
  PUT m1=;  
  m2 = LGAMMA(-2.2);  
  PUT m2=;  
RUN;
```

This produces the following output:

```
g1=5.8074187347  
g2=0  
g3=-0.10817481  
g4=0.7873750833  
  
m1=.  
m2=.
```

The last two examples return a missing value because the argument is out of range, or because of an overflow.

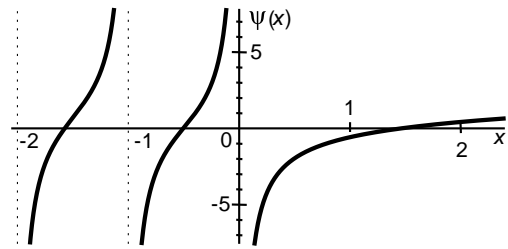
DIGAMMA

Returns the Digamma function of a numeric argument.

➤ **DIGAMMA** ➤ **(x)** ➤

The Digamma function is the logarithmic derivative of the Gamma function, see [GAMMA](#) (page 1813):

$$\psi(x) = \frac{d}{dx} \log \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the Gamma function.

Restriction: cannot be within 10^{-12} of a negative integer or zero

If the argument is out of range, a missing value is returned.

Basic examples

In these examples, the Digamma function of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = DIGAMMA(1);
  PUT g1;
  g2 = DIGAMMA(1.461632145);
  PUT g2;
  g3 = DIGAMMA(0.003);
  PUT g3;
  g4 = DIGAMMA(-0.3);
  PUT g4;
  g5 = DIGAMMA(-1.5);
  PUT g5;
  g6 = DIGAMMA(-2.7);
  PUT g6;
RUN;
```

This produces the following output:

```
g1=-0.577215665
g2=3.061493e-11
g3=-333.905625
g4=2.1133097796
g5=0.7031566406
g6=-1.115347129
```

The second example takes the positive root of the Digamma function as an argument, rounded to nine decimals. The result is nearly zero, as expected.

Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;
  m1 = DIGAMMA(0);
  PUT m1=;
  m2 = DIGAMMA(1e-13);
  PUT m2=;
  m3 = DIGAMMA(-2);
  PUT m3=;
RUN;
```

This produces the following output:

```
m1=.
m2=.
m3=.
```

The above examples return a missing value because the argument is nearing a function pole.

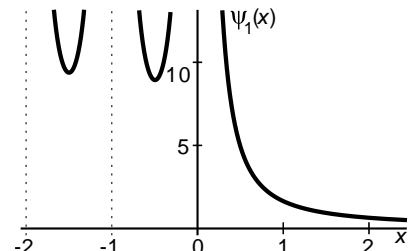
TRIGAMMA

Returns the Trigamma function of a numeric argument.

➤ **TRIGAMMA** ➤ (*x*) ➤

The Trigamma function is the second logarithmic derivative of the Gamma function, see [GAMMA](#) (page 1813):

$$\psi_1(x) = \frac{d^2}{dx^2} \log \Gamma(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the Gamma function.

Restriction: cannot be within 10^{-12} of a negative integer or zero

If the argument is out of range, a missing value is returned.

Examples

In these examples, the Trigamma function of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = TRIGAMMA(1.3);  
  PUT g1=;  
  g2 = TRIGAMMA(0.003);  
  PUT g2=;  
  g3 = TRIGAMMA(-0.3);  
  PUT g3=;  
  g4 = TRIGAMMA(-1.5);  
  PUT g4=;  
  g5 = TRIGAMMA(-2.7);  
  PUT g5=;  
  g6 = TRIGAMMA(1e200);  
  PUT g6=;  
  m1 = TRIGAMMA(0);  
  PUT m1=;  
  m2 = TRIGAMMA(1e-13);  
  PUT m2=;  
  m3 = TRIGAMMA(-2);  
  PUT m3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=1.134253435  
g2=111112.74886  
g3=13.945160268  
g4=9.3792466451  
g5=14.769375846  
g6=0  
m1=.  
m2=.  
m3=.
```

In the last example `TRIGAMMA` returns zero when a large value in the argument causes an underflow. The last three examples return a missing value because the argument is nearing a function pole.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

BETA

Returns the Beta function of two numeric arguments, or the Euler integral of the first kind.

➤ **BETA** ➤ (*x* , *y*) ➤

The Beta function is defined via a convergent improper integral:

$$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

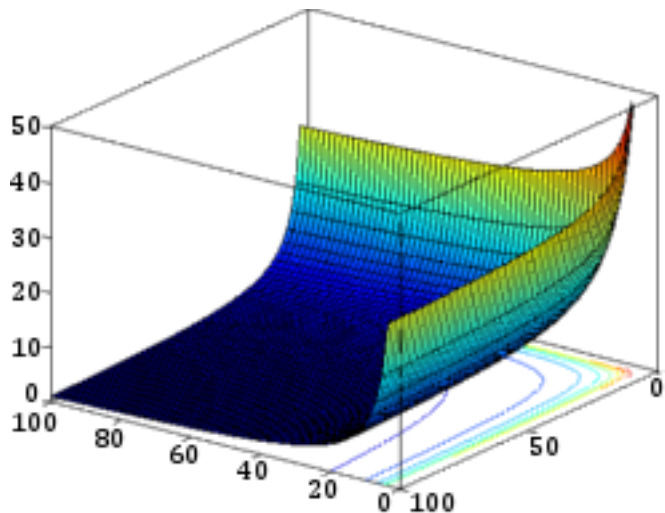
The Beta function is related to the Gamma function, see [GAMMA](#) (page 1813):

$$B(x, y) = \frac{\Gamma(x) \Gamma(y)}{\Gamma(x+y)}$$

$$\Gamma(x) \Gamma(y) = B(x, y) \Gamma(x+y)$$

For integers, the Beta function can be expressed with factorials (see [FACT](#) (page 1811)):

$$B(x, y) = \frac{(x-1)! (y-1)!}{(x+y-1)!}$$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The X coordinate value.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

y

Type: Numeric

The Y coordinate value.

Restriction: $y > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the Beta function of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
g1 = BETA(0.3,0.5);  
PUT g1=;  
g2 = BETA(6.8,2.2);  
PUT g2=;  
g3 = BETA(701.54,710.3);  
PUT g3=;  
m1 = BETA(0,0);  
PUT m1=;  
m2 = BETA(-2.2,1);  
PUT m2=;  
m3 = BETA(1e306,1);  
PUT m3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=4.554443088  
g2=0.0135704815  
g3=0  
m1=.  
m2=.  
m3=.
```

The last three examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

LOGBETA

Returns the natural logarithm of the Beta function of two numeric arguments.

➤ **LOGBETA** ➤ (*x* , *y*) ➤

Calculates the natural logarithm of the Beta function, see *BETA* [↗](#) (page 1819):

$$\log B(x, y)$$

Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The X coordinate value.

Restriction: $x > 0$

If the argument is out of range, a missing value is returned.

y

Type: Numeric

The Y coordinate value.

Restriction: $y > 0$

If the argument is out of range, a missing value is returned.

Examples

In these examples, the natural logarithm of the Beta function of the arguments is returned. The results are written to the log.

```
DATA _NULL_ ;  
  g1 = LOGBETA(0.3,0.5) ;  
  PUT g1= ;  
  g2 = LOGBETA(6.8,2.2) ;  
  PUT g2= ;  
  g3 = LOGBETA(701.54,710.3) ;  
  PUT g3= ;  
  m1 = LOGBETA(0,0) ;  
  PUT m1= ;  
  m2 = LOGBETA(-2.2,1) ;  
  PUT m2= ;  
  m3 = LOGBETA(1e306,1) ;  
  PUT m3= ;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
g1=1.5161032593  
g2=-4.299858325  
g3=-980.5997813  
m1= .  
m2= .  
m3= .
```

The last three examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

JBESSEL

Returns the Bessel function of the first kind for two numeric arguments.

⇒ **JBESSEL** ⇒ (*alpha* , *x*) ⇒

Calculates the Bessel function of the first kind:

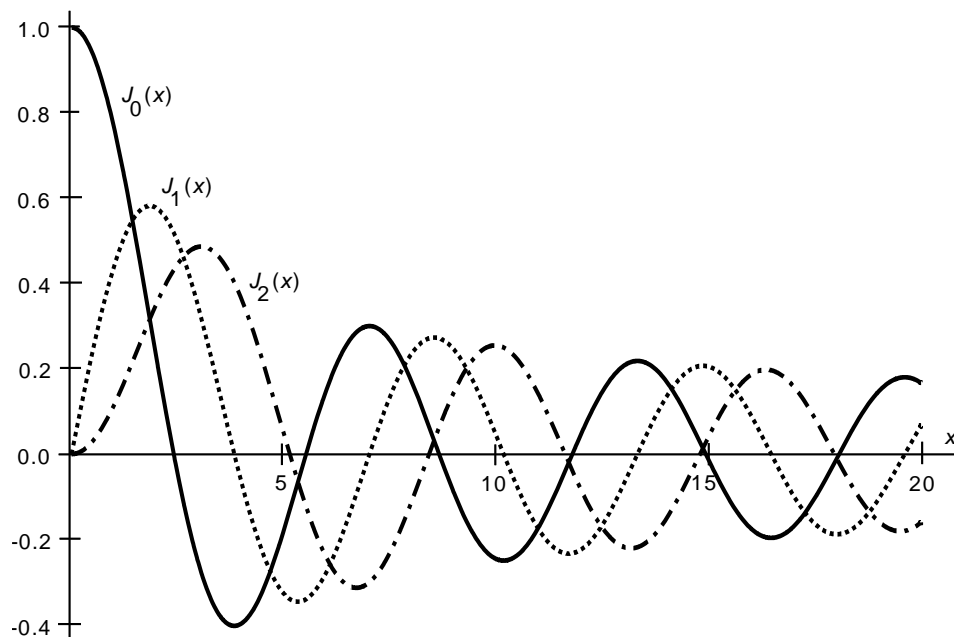
$$J_{\alpha}(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

for all integer m , using the factorial $m!$ (see [FACT](#) (page 1811)) and the Gamma function $\Gamma(t)$ (see [GAMMA](#) (page 1813)).

The above equation defines a family of functions parametrised by the value of α known as the *order* of the function. The functions represent a group of solutions of Bessel's differential equation:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

The plot below illustrates $J_{\alpha}(x)$ functions of orders 0, 1 and 2.



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

alpha

Type: Numeric

The order of the function.

Restriction: $\alpha \geq 0$

If the argument is out of range, a missing value is returned.

x

Type: Numeric

The point at which to calculate the Bessel function of the first kind.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

Basic examples

In these examples, the Bessel function of the first kind of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = JBESSEL(0.3,0.5);  
  PUT g1=;  
  g2 = JBESSEL(6.8,2.2);  
  PUT g2=;  
  g3 = JBESSEL(0,0);  
  PUT g3=;  
RUN;
```

This produces the following output:

```
g1=0.7002604885  
g2=0.0004841404  
g3=1
```

Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;  
  m1 = JBESSEL(2.2,-1);  
  PUT m1=;  
  m2 = JBESSEL(-2.2,1);  
  PUT m2=;  
  m3 = JBESSEL(200,2.3);  
  PUT m3=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
m1=.  
m2=.  
m3=.
```

The above examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

IBESSEL

Returns the modified Bessel function of the first kind for two numeric arguments, scaled or unscaled.

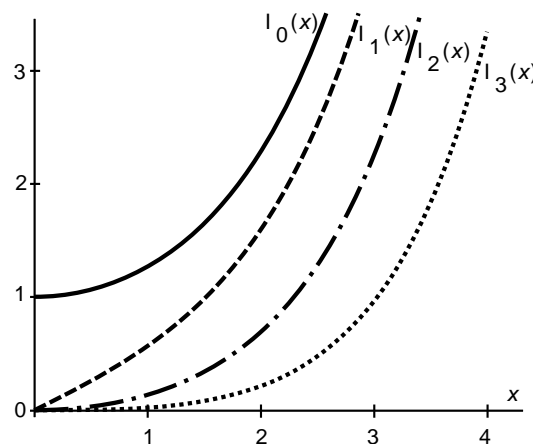
⇒ **IBESSEL** (**alpha** , **x** , **scale**) ⇐

Calculates the modified Bessel function of the first kind:

$$I_{\alpha}(x) = \sum_{m=0}^{\infty} \frac{1}{m! \Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}$$

$$\begin{cases} \text{if } scale=1 & \text{return } e^{-x} I_{\alpha}(x) \\ \text{otherwise} & \text{return } I_{\alpha}(x) \end{cases}$$

for all integer m , using the factorial $m!$ (see [FACT](#) (page 1811)) and the Gamma function $\Gamma(t)$ (see [GAMMA](#) (page 1813)).



$I_{\alpha}(x)$ functions of orders 0, 1, 2 and 3

The above equation defines a family of functions parametrised by the value of α known as the *order* of the function. The functions represent a group of solutions of Bessel's differential equation:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - \alpha^2) y = 0$$

Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

alpha

Type: Numeric

The order of the function.

Restriction: $alpha \geq 0$

If the argument is out of range, a missing value is returned.

x

Type: Numeric

The point at which to calculate the modified Bessel function of the first kind.

Restriction: $x \geq 0$

If the argument is out of range, a missing value is returned.

scale

Type: Numeric

The scale of the function.

Restriction: $0 \leq scale \leq 1$

If the argument is out of range, a missing value is returned.

Basic examples

In these examples, the modified Bessel function of the first kind of the arguments is returned. The results are written to the log.

```
DATA _NULL_;  
  g1 = IBESSEL(0.3,0.5,0);  
  PUT g1=;  
  g2 = IBESSEL(0.3,0.5,1);  
  PUT g2=;  
  g3 = IBESSEL(6.3,22.2,0);  
  PUT g3=;  
  g4 = IBESSEL(6.3,22.2,1);  
  PUT g4=;  
  g5 = IBESSEL(0,0,0);  
  PUT g5=;  
  g6 = IBESSEL(0,0,1);  
  PUT g6=;  
RUN;
```

This produces the following output:

```
g1=0.7709517346  
g2=0.4676058642  
g3=150339241.87  
g4=0.0343348265  
g5=1  
g6=1
```


Examples — returning missing values

The following examples illustrate some of the conditions that cause the function to return a missing value. The results are written to the log.

```
DATA _NULL_;
  m1 = IBESSEL(2.2,-1,0);
  PUT m1=;
  m2 = IBESSEL(-2.2,1,0);
  PUT m2=;
  m3 = IBESSEL(-2.2,1,2);
  PUT m3=;
  m4 = IBESSEL(100,1e200,0);
  PUT m4=;
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
m1=.
m2=.
m3=.
m4=.
```

The above examples return a missing value because the argument is out of range, or because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

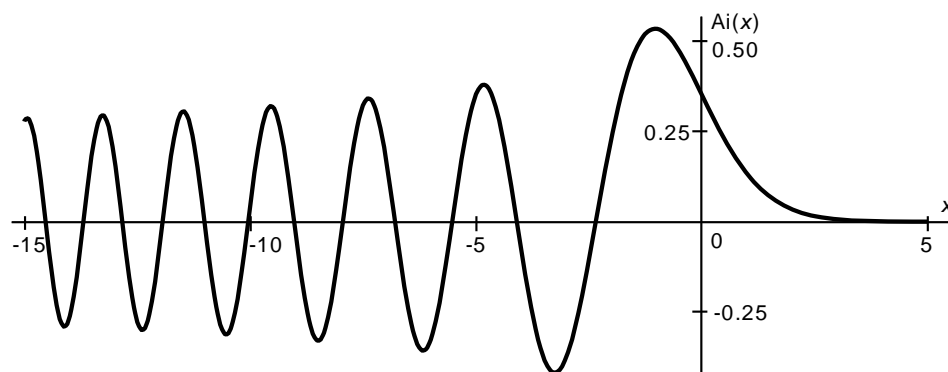
AIRY

Returns the Airy function of the first kind for a numeric argument.

⇒ **AIRY** (*x*) ⇐

The Airy function is defined via a convergent improper integral:

$$\text{Ai}(x) = \frac{1}{\pi} \int_0^{\infty} \cos\left(\frac{t^3}{3} + xt\right) dt$$



The $Ai(x)$ function is a solution to the Airy differential equation:

$$\frac{d^2y}{dx^2} - xy = 0$$

Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the Airy function of the first kind.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the Airy function of the first kind of the argument is returned. The results are written to the log.

```
DATA _NULL_;  
  a1 = AIRY(1.3);  
  PUT a1=;  
  a2 = AIRY(0);  
  PUT a2=;  
  a3 = AIRY(104);  
  PUT a3=;  
  a4 = AIRY(105);  
  PUT a4=;  
  a5 = AIRY(-1e205);  
  PUT a5=;  
  m1 = AIRY(-1e206);  
  PUT m1=;  
  m2 = AIRY( 1e206);  
  PUT m2=;  
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
a1=0.0934746658  
a2=0.3550280539  
a3=0.74488e-308  
a4=0  
a5=-2.45245e-52  
m1=.  
m2=.
```

The last two examples return a missing value because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

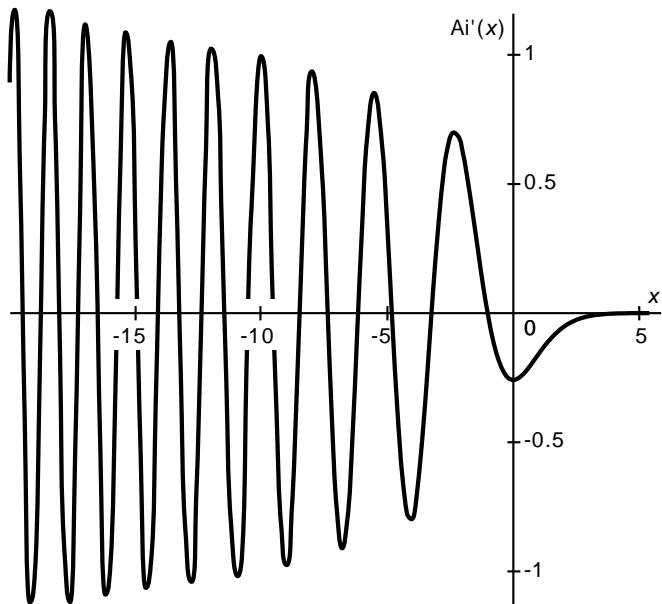
DAIRY

Returns the first derivative of the Airy function of the first kind for a numeric argument.

⇒ **DAIRY** (*x*) ⇒

Calculates the first derivative of the Airy function of the first kind, see [AIRY](#) (page 1827):

$Ai'(x)$



Return type: Numeric

If an overflow occurs as a result of the operation, a missing value is returned.

x

Type: Numeric

The point at which to calculate the first derivative of the Airy function of the first kind.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the first derivative of the Airy function of the first kind of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  a1 = DAIRY(1.3);
  PUT a1=;
  a2 = DAIRY(0);
  PUT a2=;
  a3 = DAIRY(104);
  PUT a3=;
  a4 = DAIRY(105);
  PUT a4=;
  a5 = DAIRY(-1e205);
  PUT a5=;
  m1 = DAIRY(-1e206);
  PUT m1=;
  m2 = DAIRY( 1e206);
  PUT m2=;
RUN;
```

On a Windows 64-bit computer, this produces the following output:

```
a1=-0.120333866
a2=-0.258819404
a3=-7.5981e-308
a4=0
a5=-6.365001e50
m1=.
m2=.
```

The last two examples return a missing value because of an overflow.

Note:

Other platforms might have different boundary values as determined by their architecture, resulting in different outcomes in the examples.

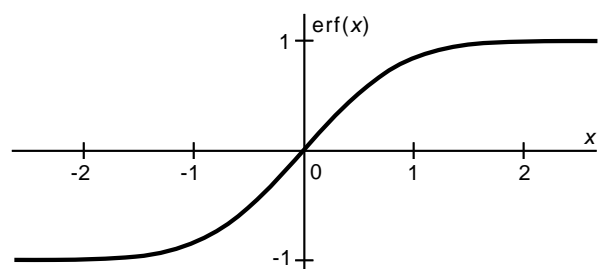
ERF

Returns the error function of a numeric argument.

⇒ **ERF** ⇒ (x) ⇒

Calculates the error function of the argument:

$$\operatorname{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^x \exp(-t^2) dt$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the error function.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the error function of the argument is returned. The results are written to the log.

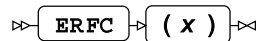
```
DATA _NULL_;  
  g1 = ERF(-3.3);  
  PUT g1=;  
  g2 = ERF(0.003);  
  PUT g2=;  
RUN;
```

This produces the following output:

```
g1=-0.999996942  
g2=0.0033851273
```

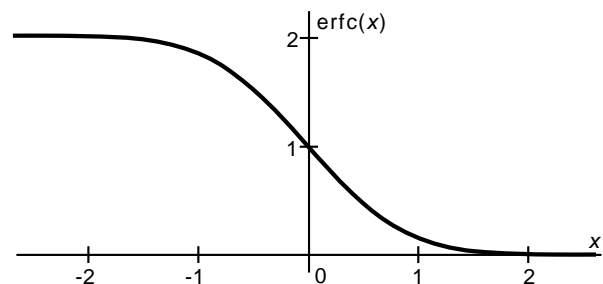
ERFC

Returns the complementary error function of a numeric argument.



Calculates the complementary error function based on the error function, see [ERF](#) (page 1830).

$$\text{erfc}(x) = 1 - \text{erf}(x)$$



Return type: Numeric

x

Type: Numeric

The point at which to calculate the complementary error function.

If the argument contains a missing value, a missing value is returned.

Examples

In these examples, the complementary error function of the argument is returned. The results are written to the log.

```
DATA _NULL_;
  g1 = ERFC(-3.3);
  PUT g1=;
  g2 = ERFC(0.003);
  PUT g2=;
RUN;
```

This produces the following output:

```
g1=1.9999969423
g2=0.9966148727
```

Value counts

Return counts for lists of values.

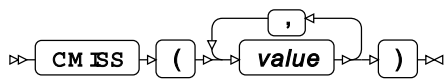
Examples in this section use a list of values containing a mix of numeric, character and missing values:

1, -2, 3, ., "a", "4", ""

CMISS ↗	1832
Returns the number of missing values in a list of character or numeric values.	
NMISS ↗	1833
Returns the number of missing values in a list of numeric values.	
N ↗	1834
Returns the number of non-missing values in a list of numeric values.	

CMISS

Returns the number of missing values in a list of character or numeric values.



Defined for character and numeric values.

Return type: Numeric

value

Type: Character or numeric value

The value to be evaluated.

Example

In this example, the number of missing values in the argument list is returned. The result is written to the log.

```
DATA _NULL_;  
  c = CMISS(1,-2,3,., "a", "4", "");  
  PUT c=;  
RUN;
```

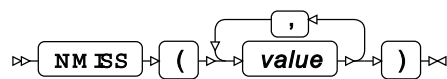
This produces the following output:

```
c=2
```

The output is 2, as only the fourth (.) and last (" ") values are considered missing.

NMISS

Returns the number of missing values in a list of numeric values.



Defined for numeric values only.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Example

In this example, the number of missing values in the argument list is returned. The result is written to the log.

```
DATA _NULL_;  
  n = NMISS(1,-2,3,., "a", "4", "");  
  PUT n=;  
RUN;
```

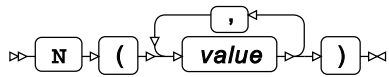
This produces the following output:

```
n=3
```

The sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last (" ") values are considered missing.

N

Returns the number of non-missing values in a list of numeric values.



Defined for numeric values only.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Example

In this example, the number of non-missing values in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  n = N(1, -2, 3, ., "a", "4", "");
  PUT n=;
RUN;
```

This produces the following output:

```
n=4
```

The sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last ("") values are considered missing.

Minimum and maximum values

Operate on the minimum and maximum in a list of values.

Examples in this section use a list of values containing a mix of numeric, character and missing values:

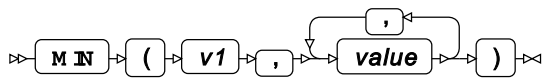
1, -2, 3, ., "a", "4", ""

MIN ↗	1835
Returns the minimum value in a list of numeric values.	
MAX ↗	1836
Returns the maximum value in a list of numeric values.	
RANGE ↗	1836
Returns the difference between the maximum and minimum value in a list of numeric values.	

SMALLEST ↗	1837
Returns the <i>n</i> -th smallest value in a list of numeric values.	
LARGEST ↗	1838
Returns the <i>n</i> -th largest value in a list of numeric values.	
ORDINAL ↗	1839
Returns the <i>n</i> -th smallest value in a list of numeric or missing values.	

MIN

Returns the minimum value in a list of numeric values.



Requires at least two arguments. Calculates the minimum value of non-missing values

Return type: Numeric

If all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Example

In this example, the minimum value in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  m = MIN(1,-2,3,., "a", "4", "");
  PUT m=;
RUN;
```

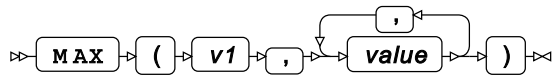
This produces the following output:

```
m=-2
```

The sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last ("") values are considered missing.

MAX

Returns the maximum value in a list of numeric values.



Requires at least two arguments. Calculates the maximum value of non-missing values.

Return type: Numeric

If all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Example

In this example, the maximum value in the argument list is returned. The result is written to the log.

```
DATA _NULL_;  
m = MAX(1, -2, 3, ., "a", "4", "");  
PUT m=;  
RUN;
```

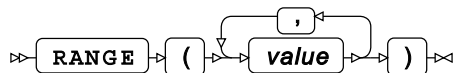
This produces the following output:

```
m=4
```

The output is 4, as the sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last ("") values are considered missing.

RANGE

Returns the difference between the maximum and minimum value in a list of numeric values.



Calculates the difference between the maximum and minimum value of non-missing values.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the difference between the maximum and minimum value in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  r = RANGE(1, -2, 3, ., "a", "4", "");
  PUT r=;
RUN;
```

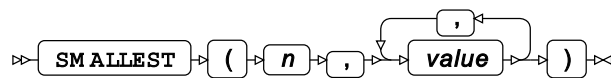
This produces the following output:

```
r=6
```

The calculation is as follows: $4 - (-2) = 6$.

SMALLEST

Returns the n -th smallest value in a list of numeric values.



Calculates the n -th smallest value of non-missing values.

Return type: Numeric

n

Type: Numeric

The ordinal position in the rank from smallest to largest values.

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the n -th smallest value in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  s = SMALLEST(2, 1, -2, 3, ., "a", "4", "");
  PUT s=;
RUN;
```

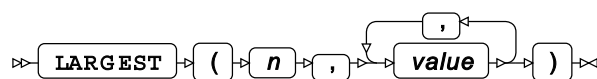
This produces the following output:

```
s=1
```

The second smallest value is 1, as the sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last (" ") values are considered missing.

LARGEST

Returns the n -th largest value in a list of numeric values.



Calculates the n -th largest value of non-missing values

Return type: Numeric

n

Type: Numeric

The ordinal position in the rank from smallest to largest values.

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the n -th largest value in the argument list is returned. The result is written to the log.

```
DATA _NULL_;
  s = LARGEST(2, 1, -2, 3, ., "a", "4", "");
  PUT s=;
RUN;
```

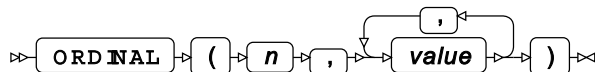
This produces the following output:

```
s=3
```

The second largest value is 3, as the sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last (" ") values are considered missing.

ORDINAL

Returns the n -th smallest value in a list of numeric or missing values.



Defined for numeric and missing values. Character values that cannot be converted into numbers, are considered missing.

Returns the n -th smallest value in the list. Missing values are considered smaller than any non-missing values.

Return type: Numeric

n

Type: Numeric

The ordinal position in the rank from smallest to largest values.

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the n -th smallest value in the argument list is returned. The results are written to the log.

```

DATA _NULL_;
o1 = ORDINAL(1, 1,-2,3,., "a", "4", " ");
PUT o1=;
o2 = ORDINAL(2, 1,-2,3,., "a", "4", " ");
PUT o2=;
o3 = ORDINAL(3, 1,-2,3,., "a", "4", " ");
PUT o3=;
o4 = ORDINAL(4, 1,-2,3,., "a", "4", " ");
PUT o4=;
RUN;
  
```

This produces the following output:

```
o1=.
o2=.
o3=.
o4=-2
```

The first value in the argument list is the function parameter *n*. The remainder forms the list of values to process.

The sixth value ("4") has been converted into a number, but the fourth (.), the fifth ("a") and last (" ") values are considered missing. The function returns a missing value (.) as the first, second and third smallest value in the list (o1, o2 and o3, respectively). The fourth smallest value returned (o4) is -2, the smallest non-missing value in this example.

Percentile-based calculations

Calculate percentile-based values.

Examples in this section use two similar lists of values. The first list contains values close to each other, while the second list contains the same values plus a significantly larger outlier:

1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", ""
1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15

PCTL methods ↗	1840
Calculate percentile-based values with different methods.	
MEDIAN ↗	1853
PCTL5 (50, x_1, \dots, x_n)	
Returns the median of a list of numeric values.	
MAD ↗	1853
MEDIAN ($ x_1 - \text{MEDIAN}(x_1, \dots, x_n) , \dots, x_n - \text{MEDIAN}(x_1, \dots, x_n) $)	
Returns the median absolute deviation from the median of a list of numeric values.	
IQR ↗	1854
PCTL5 (75, x_1, \dots, x_n) – PCTL5 (25, x_1, \dots, x_n)	
Returns the inter-quartile range of a list of numeric values.	

PCTL methods

Calculate percentile-based values with different methods.

There are five methods for percentile calculation implemented by the correspondingly numbered functions PCTL1 through PCTL5.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q depending on the method:

$$\text{method 4} \quad q = p(n+1) / 100$$

$$\text{methods 1, 2, 3 and 5} \quad q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\lfloor q \rfloor = \text{floor}(q)$$

$$i = \lfloor q \rfloor - 1$$

$$h = q - \lfloor q \rfloor$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$\text{methods 1 and 4} \quad (1-h)x_i + hx_{i+1}$$

$$\begin{aligned} \text{method 2} \quad & \text{if } h < 0.5 && \text{return } x_i \\ & \text{if } h > 0.5 && \text{return } x_{i+1} \\ & \text{if } h = 0.5 \text{ and } i \text{ is even} && \text{return } x_i \\ & \text{if } h = 0.5 \text{ and } i \text{ is odd} && \text{return } x_{i+1} \end{aligned}$$

$$\begin{aligned} \text{method 3} \quad & \text{if } h = 0 && \text{return } x_i \\ & \text{otherwise} && \text{return } x_{i+1} \end{aligned}$$

$$\begin{aligned} \text{method 5} \quad & \text{if } h = 0 && \text{return } \frac{1}{2}(x_i + x_{i+1}) \\ & \text{otherwise} && \text{return } x_{i+1} \end{aligned}$$

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section.

```
DATA EXAMPLE;
DO i = 0 TO 100 BY 25;
  p1 = PCTL1(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  p2 = PCTL2(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  p3 = PCTL3(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  p4 = PCTL4(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  p5 = PCTL5(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  OUTPUT;
END;

DO i = 0 TO 100 BY 25;
  p1 = PCTL1(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  p2 = PCTL2(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  p3 = PCTL3(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  p4 = PCTL4(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  p5 = PCTL5(i, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  OUTPUT;
END;
RUN;
```

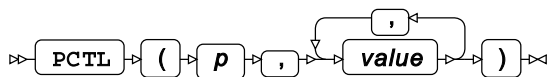
The contents of the `EXAMPLE` dataset are as follows:

	value list 1					value list 2				
percentile	PCTL1	PCTL2	PCTL3	PCTL4	PCTL5	PCTL1	PCTL2	PCTL3	PCTL4	PCTL5
0	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
25	1.025	1	1.1	1.05	1.1	1.05	1	1.1	1.075	1.1
50	1.25	1.2	1.3	1.3	1.3	1.3	1.3	1.3	1.35	1.35
75	1.475	1.5	1.5	2.25	1.5	2.25	3	3	3.25	3
100	4	4	4	4	4	10 ¹⁵	10 ¹⁵	10 ¹⁵	10 ¹⁵	10 ¹⁵

PCTL ↗	1842
Returns a percentile a list of numeric values according to the default method. This function is an alias of PCTL5.	
PCTL1 ↗	1844
Returns a percentile of a list of numeric values according to method 1.	
PCTL2 ↗	1846
Returns a percentile of a list of numeric values according to method 2.	
PCTL3 ↗	1847
Returns a percentile of a list of numeric values according to method 3.	
PCTL4 ↗	1849
Returns a percentile of a list of numeric values according to method 4.	
PCTL5 ↗	1851
Returns a percentile a list of numeric values according to method 5. This function is an alias of PCTL.	

PCTL

Returns a percentile a list of numeric values according to the default method. This function is an alias of PCTL5.



Calculates their specified percentile p of non-missing values calculated with the default method – method 5.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\begin{aligned} \lfloor q \rfloor &= \text{floor}(q) \\ i &= \lfloor q \rfloor - 1 \\ h &= q - \lfloor q \rfloor \end{aligned}$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$\begin{aligned} \text{if } h=0 & \quad \text{return } \frac{1}{2}(x_i + x_{i+1}) \\ \text{otherwise} & \quad \text{return } x_{i+1} \end{aligned}$$

Return type: Numeric

p

Type: Numeric

The percentile to find.

Restriction: $0 \leq p \leq 100$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0   = PCTL( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p0=;
  p25  = PCTL( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p25=;
  p50  = PCTL( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p50=;
  p75  = PCTL( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p75=;
  p100 = PCTL(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p100=;

  q0   = PCTL( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q0=;
  q25  = PCTL( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q25=;
  q50  = PCTL( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q50=;
  q75  = PCTL( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q75=;
```

```
q100 = PCTL(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
PUT q100=;
RUN;
```

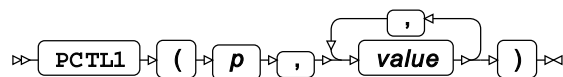
This produces the following output:

```
p0=-2
p25=1.1
p50=1.3
p75=1.5
p100=4

q0=-2
q25=1.1
q50=1.35
q75=3
q100=1e15
```

PCTL1

Returns a percentile of a list of numeric values according to method 1.



Calculates their specified percentile p of non-missing values calculated with method 1.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\lfloor q \rfloor = \text{floor}(q)$$

$$i = \lfloor q \rfloor - 1$$

$$h = q - \lfloor q \rfloor$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$(1-h)x_i + hx_{i+1}$$

Return type: Numeric

p

Type: Numeric

The percentile to find.

Restriction: $0 \geq p \geq 100$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0   = PCTL1( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p0=;
  p25  = PCTL1( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p25=;
  p50  = PCTL1( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p50=;
  p75  = PCTL1( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p75=;
  p100 = PCTL1(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p100=;

  q0   = PCTL1( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q0=;
  q25  = PCTL1( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q25=;
  q50  = PCTL1( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q50=;
  q75  = PCTL1( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q75=;
  q100 = PCTL1(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q100=;
RUN;
```

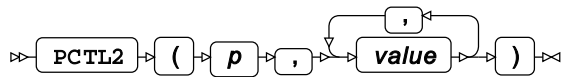
This produces the following output:

```
p0=-2
p25=1.025
p50=1.25
p75=1.475
p100=4

q0=-2
q25=1.05
q50=1.3
q75=2.25
q100=1e15
```

PCTL2

Returns a percentile of a list of numeric values according to method 2.



Calculates their specified percentile p of non-missing values calculated with method 2.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\begin{aligned} \lfloor q \rfloor &= \text{floor}(q) \\ i &= \lfloor q \rfloor - 1 \\ h &= q - \lfloor q \rfloor \end{aligned}$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$\begin{aligned} \text{if } h < 0.5 & \quad \text{return } x_i \\ \text{if } h > 0.5 & \quad \text{return } x_{i+1} \\ \text{if } h = 0.5 \text{ and } i \text{ is even} & \quad \text{return } x_i \\ \text{if } h = 0.5 \text{ and } i \text{ is odd} & \quad \text{return } x_{i+1} \end{aligned}$$

Return type: Numeric

p

Type: Numeric

The percentile to find.

Restriction: $0 \leq p \leq 100$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0    = PCTL2( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p0=;
  p25   = PCTL2( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p25=;
  p50   = PCTL2( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p50=;
  p75   = PCTL2( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p75=;
  p100  = PCTL2(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p100=;

  q0    = PCTL2( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q0=;
  q25   = PCTL2( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q25=;
  q50   = PCTL2( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q50=;
  q75   = PCTL2( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q75=;
  q100  = PCTL2(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q100=;
RUN;
```

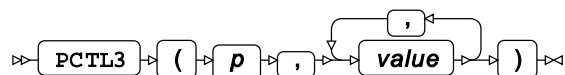
This produces the following output:

```
p0=-2
p25=1
p50=1.2
p75=1.5
p100=4

q0=-2
q25=1
q50=1.3
q75=3
q100=1e15
```

PCTL3

Returns a percentile of a list of numeric values according to method 3.



Calculates their specified percentile p of non-missing values calculated with method 3.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\begin{aligned} \lfloor q \rfloor &= \text{floor}(q) \\ i &= \lfloor q \rfloor - 1 \\ h &= q - \lfloor q \rfloor \end{aligned}$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

```
if h = 0      return  $x_i$ 
otherwise return  $x_{i+1}$ 
```

Return type: Numeric

p

Type: Numeric

The percentile to find.

Restriction: $0 \leq p \leq 100$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0 = PCTL3( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "" );
  PUT p0=;
  p25 = PCTL3( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "" );
  PUT p25=;
  p50 = PCTL3( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "" );
  PUT p50=;
  p75 = PCTL3( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "" );
  PUT p75=;
  p100 = PCTL3(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "" );
  PUT p100=;

  q0 = PCTL3( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15 );
  PUT q0=;
```

```

q25 = PCTL3( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
PUT q25=;
q50 = PCTL3( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
PUT q50=;
q75 = PCTL3( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
PUT q75=;
q100 = PCTL3(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
PUT q100=;
RUN;

```

This produces the following output:

```

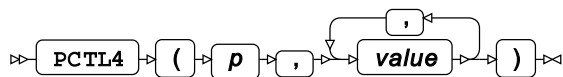
p0=-2
p25=1.1
p50=1.3
p75=1.5
p100=4

q0=-2
q25=1.1
q50=1.3
q75=3
q100=1e15

```

PCTL4

Returns a percentile of a list of numeric values according to method 4.



Calculates their specified percentile p of non-missing values calculated with method 4.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p(n+1) / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\begin{aligned} \lfloor q \rfloor &= \text{floor}(q) \\ i &= \lfloor q \rfloor - 1 \\ h &= q - \lfloor q \rfloor \end{aligned}$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$(1-h)x_i + hx_{i+1}$$

Return type: Numeric

p**Type:** Numeric

The percentile to find.

Restriction: $0 \geq p \geq 100$

If the argument is out of range, a missing value is returned.

value**Type:** Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0    = PCTL4( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p0=;
  p25   = PCTL4( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p25=;
  p50   = PCTL4( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p50=;
  p75   = PCTL4( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p75=;
  p100  = PCTL4(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p100=;

  q0    = PCTL4( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q0=;
  q25   = PCTL4( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q25=;
  q50   = PCTL4( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q50=;
  q75   = PCTL4( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q75=;
  q100  = PCTL4(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q100=;
RUN;
```

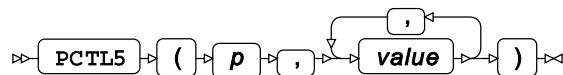

This produces the following output:

```
p0=-2
p25=1.05
p50=1.3
p75=2.25
p100=4

q0=-2
q25=1.075
q50=1.35
q75=3.25
q100=1e15
```

PCTL5

Returns a percentile a list of numeric values according to method 5. This function is an alias of PCTL.



Calculates their specified percentile p of non-missing values calculated with method 5.

Percentile is calculated as follows:

1. Non-missing values are sorted into ascending order allowing the identification of two adjacent values x_i and x_{i+1} between which falls the specified percentile. The index i is calculated using a helper variable q :

$$q = p \cdot n / 100$$

2. Using q , index i and a helper value h needed in the next step, are calculated based on the *floor* of q , that is, the value of q rounded down to the nearest whole number:

$$\begin{aligned} \lfloor q \rfloor &= \text{floor}(q) \\ i &= \lfloor q \rfloor - 1 \\ h &= q - \lfloor q \rfloor \end{aligned}$$

3. The percentile is calculated based on i th and $(i+1)$ st values from the sorted list:

$$\begin{aligned} \text{if } h = 0 & \quad \text{return } \frac{1}{2}(x_i + x_{i+1}) \\ \text{otherwise} & \quad \text{return } x_{i+1} \end{aligned}$$

Return type: Numeric

p

Type: Numeric

The percentile to find.

Restriction: $0 \leq p \leq 100$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In the following example, the 0-, 25-, 50-, 75- and 100-percentile are calculated for the two value lists used in this section. The results are written to the log.

```
DATA _NULL_;
  p0    = PCTL5( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p0=;
  p25   = PCTL5( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p25=;
  p50   = PCTL5( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p50=;
  p75   = PCTL5( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p75=;
  p100  = PCTL5(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT p100=;

  q0    = PCTL5( 0, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q0=;
  q25   = PCTL5( 25, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q25=;
  q50   = PCTL5( 50, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q50=;
  q75   = PCTL5( 75, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q75=;
  q100  = PCTL5(100, 1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q100=;
RUN;
```

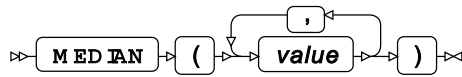
This produces the following output:

```
p0=-2
p25=1.1
p50=1.3
p75=1.5
p100=4

q0=-2
q25=1.1
q50=1.35
q75=3
q100=1e15
```

MEDIAN

Returns the median of a list of numeric values.



Calculates the median of non-missing values calculated as a 50-percentile using function `PCTL5`, see [PCTL5](#) (page 1851):

$$\text{PCTL5}(50, x_1, \dots, x_n)$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the median of the arguments is returned. The results are written to the log.

```

DATA _NULL_;
  m1 = MEDIAN(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "");
  PUT m1;
  m2 = MEDIAN(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15);
  PUT m2;
RUN;
  
```

This produces the following output:

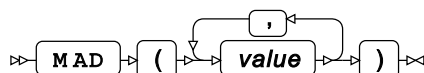
```

m1=1.3
m2=1.35
  
```

The outlier in the second value list has only a small impact on the median.

MAD

Returns the median absolute deviation from the median of a list of numeric values.



Calculates the median absolute deviation from the median of non-missing values (see [MEDIAN](#) (page 1853)):

$$\text{MEDIAN}(|x_1 - \text{MEDIAN}(x_1, \dots, x_n)|, \dots, |x_n - \text{MEDIAN}(x_1, \dots, x_n)|)$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the median absolute deviation from the median of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  d1 = MAD(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "");
  PUT d1=;
  d2 = MAD(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15);
  PUT d2=;
RUN;
```

This produces the following output:

```
d1=0.2
d2=0.3
```

This function uses median calculations shown in variables m1 and m2:

$$m_1 = \text{MEDIAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) = 1.3$$

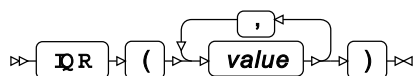
$$m_2 = \text{MEDIAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) = 1.35$$

$$d_1 = \text{MEDIAN}\left(\begin{array}{l} |1.1 - 1.3|, |1.2 - 1.3|, |1.3 - 1.3|, |1.4 - 1.3|, \\ |1.5 - 1.3|, |1 - 1.3|, |-2 - 1.3|, |3 - 1.3|, |4 - 1.3| \end{array}\right) = 0.2$$

$$d_2 = \text{MEDIAN}\left(\begin{array}{l} |1.1 - 1.35|, |1.2 - 1.35|, |1.3 - 1.35|, |1.4 - 1.35|, |1.5 - 1.35|, \\ |1 - 1.35|, |-2 - 1.35|, |3 - 1.35|, |4 - 1.35|, |10^{15} - 1.35| \end{array}\right) = 0.3$$

IQR

Returns the inter-quartile range of a list of numeric values.



Calculates the inter-quartile range of non-missing values calculated using function `PCTL5` (see [PCTL5](#) (page 1851)) as the difference between the 75-percentile and the 25-percentile of the value list:

$$\text{PCTL5}(75, x_1, \dots, x_n) - \text{PCTL5}(25, x_1, \dots, x_n)$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the inter-quartile range of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  q1 = IQR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT q1=;
  q2 = IQR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT q2=;
RUN;
```

This produces the following output:

```
q1=0.4
q2=1.9
```

The calculation is as follows:

$$\begin{aligned} q_1 &= \text{PCTL5}(75, 1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) \\ &\quad - \text{PCTL5}(25, 1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) = 1.5 - 1.1 = 0.4 \\ q_2 &= \text{PCTL5}(75, 1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) \\ &\quad - \text{PCTL5}(25, 1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) = 3 - 1.1 = 1.9 \end{aligned}$$

Sums and sums of squares

Calculate sums and sums of squares of a list of numeric values.

Examples in this section use a list of values containing a mix of numeric, character and missing values:

1, -2, 3, ., "a", "4", ""

[SUM](#) 1856

$$\sum_{i=1}^n x_i$$

Returns the sum of values of a list of numeric values.

SUMABS [↗](#)..... 1857

$$\sum_{i=1}^n |x_i|$$

Returns the sum of absolute values of a list of numeric values.

USS [↗](#)..... 1858

$$\sum_{i=1}^n x_i^2$$

Returns the uncorrected sum of squares of a list of numeric values.

CSS [↗](#)..... 1859

$$\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2$$

Returns the corrected sum of squares of a list of numeric values.

EUCLID [↗](#)..... 1859

$$\sqrt{\sum_{i=1}^n x_i^2}$$

Returns the Euclidean norm of a list of numeric values.

RMS [↗](#)..... 1860

$$\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}$$

Returns the root mean square of a list of numeric values.

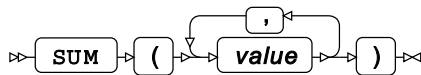
LPNORM [↗](#)..... 1861

$$\left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}$$

Returns the L_p norm of a list of numeric values.

SUM

Returns the sum of values of a list of numeric values.



Calculates the sum of non-missing values:

$$\sum_{i=1}^n x_i$$

where n is the number of non-missing values in the list.

Return type: Numeric

value**Type:** Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the sum of the arguments is returned. The result is written to the log.

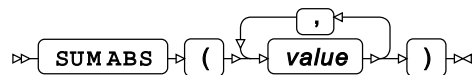
```
DATA _NULL_;
  s = SUM(1, -2, 3, ., "a", "4", "");
  PUT s=;
RUN;
```

This produces the following output:

```
s=6
```

The calculation is as follows: $1 - 2 + 3 + 4 = 6$.**SUMABS**

Returns the sum of absolute values of a list of numeric values.



Calculates the sum of absolute values of non-missing values:

$$\sum_{i=1}^n |x_i|$$

where n is the number of non-missing values in the list. The sum of absolute values is also known as the Manhattan norm.**Return type:** Numeric**value****Type:** Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the sum of absolute values of the arguments is returned. The result is written to the log.

```
DATA _NULL_;
  s = SUMABS(1,-2,3,., "a", "4", "");
  PUT s=;
RUN;
```

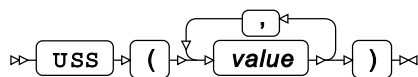
This produces the following output:

```
s=10
```

The calculation is as follows: $|1| + |-2| + |3| + |4| = 10$.

USS

Returns the uncorrected sum of squares of a list of numeric values.



Calculates the uncorrected sum of squares of non-missing values:

$$\sum_{i=1}^n x_i^2$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the uncorrected sum of squares of the arguments is returned. The result is written to the log.

```
DATA _NULL_;
  u = USS(1,-2,3,., "a", "4", "");
  PUT u=;
RUN;
```

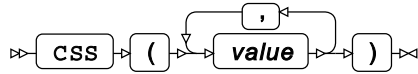
This produces the following output:

```
u=30
```

The calculation is as follows: $1^2 + (-2)^2 + 3^2 + 4^2 = 30$.

CSS

Returns the corrected sum of squares of a list of numeric values.



Calculates the corrected sum of squares of non-missing values:

$$\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the corrected sum of squares of the arguments is returned. The result is written to the log.

```
DATA _NULL_;
  c = CSS(1, -2, 3, ., "a", "4", "");
  PUT c=;
RUN;
```

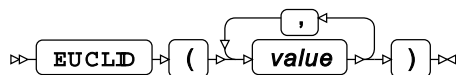
This produces the following output:

```
c=21
```

The calculation is as follows: $1^2 + (-2)^2 + 3^2 + 4^2 - \frac{1}{4}(1 - 2 + 3 + 4)^2 = 21$.

EUCLID

Returns the Euclidean norm of a list of numeric values.



Calculates the Euclidean norm of non-missing values:

$$\sqrt{\sum_{i=1}^n x_i^2}$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the Euclidean norm of the arguments is returned. The result is written to the log.

```
DATA _NULL_;
  d = EUCLID(1,-2,3,., "a", "4", "");
  PUT d=;
RUN;
```

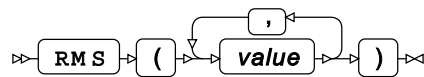
This produces the following output:

```
d=5.4772255751
```

The calculation is as follows: $\sqrt{1^2 + (-2)^2 + 3^2 + 4^2} = 5.4772255751$.

RMS

Returns the root mean square of a list of numeric values.



Calculates the root mean square of non-missing values:

$$\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}$$

where n is the number of non-missing values in the list.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the root mean square of the arguments is returned. The result is written to the log.

```
DATA _NULL_;
  r = RMS(1, -2, 3, ., "a", "4", "");
  PUT r=;
RUN;
```

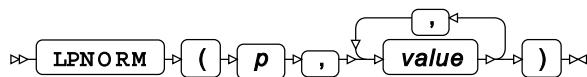
This produces the following output:

```
r=2.7386127875
```

The calculation is as follows: $\sqrt{\frac{1}{4}(1^2 + (-2)^2 + 3^2 + 4^2)} = 2.7386127875$.

LPNORM

Returns the L_p norm of a list of numeric values.



Calculates the L_p norm of non-missing values:

$$\left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}}$$

where n is the number of non-missing values in the list.

Return type: Numeric

p

Type: Numeric

The order of the L -norm.

Restriction: $p \geq 1$

If the argument is out of range, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

The first two L_p norms, L_1 or the Manhattan norm, and L_2 or the Euclidean norm, are also implemented with functions `SUMABS` and `EUCLID`, see sections [SUMABS](#) (page 1857) and [EUCLID](#) (page 1859), respectively.

Examples

In these examples, the L_p norm of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
s  = SUMABS( 1, -2, 3, ., "a", "4", "" );
PUT s=;
e  = EUCLID( 1, -2, 3, ., "a", "4", "" );
PUT e=;
l1 = LPNORM(1, 1, -2, 3, ., "a", "4", "" );
PUT l1=;
l2 = LPNORM(2, 1, -2, 3, ., "a", "4", "" );
PUT l2=;
l3 = LPNORM(3, 1, -2, 3, ., "a", "4", "" );
PUT l3=;
l4 = LPNORM(4, 1, -2, 3, ., "a", "4", "" );
PUT l4=;
RUN;
```

This produces the following output:

```
s=10
e=5.4772255751
l1=10
l2=5.4772255751
l3=4.6415888336
l4=4.3376131365
```

The calculation is as follows:

$$\begin{aligned}
 s &= l_1 = |1| + |-2| + |3| + |4| = 10 \\
 e &= l_2 = \sqrt{1^2 + (-2)^2 + 3^2 + 4^2} = 5.4772255751 \\
 l_3 &= (|1|^3 + |-2|^3 + |3|^3 + |4|^3)^{\frac{1}{3}} = 4.6415888336 \\
 l_4 &= (|1|^4 + |-2|^4 + |3|^4 + |4|^4)^{\frac{1}{4}} = 4.3376131365
 \end{aligned}$$

Mean calculations

Calculate mean values.

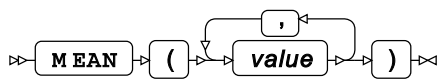
Examples in this section use two similar lists of values. The first list contains values close to each other, while the second list contains the same values plus a significantly larger outlier. In addition, the third value list contains a zero value:

```
1.1, 1.2, 1.3, 1.4, 1.5, 1, 2, 3, ., "a", "4", ""
1.1, 1.2, 1.3, 1.4, 1.5, 1, 2, 3, ., "a", "4", "", 1e15
1.1, 1.2, 1.3, 1.4, 1.5, 1, 2, 3, ., "a", "4", "", 0
```

Note:	
These value lists contain only non-negative numbers because some of the functions are not defined for negative numbers.	
MEAN ↗	1863
$\frac{1}{n} \sum_{i=1}^n x_i$ <p>Returns the arithmetic mean of a list of numeric values.</p>	
GEOMEANZ ↗	1864
$\left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}$ <p>Returns the geometric mean of a list of numeric values.</p>	
GEOMEAN ↗	1865
$\left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}$ <p>Returns the geometric mean of a list of numeric values if their spread is not too great. Returns zero otherwise.</p>	
HARMEANZ ↗	1867
$n \left(\sum_{i=1}^n x_i^{-1} \right)^{-1}$ <p>Returns the harmonic mean of a list of numeric values.</p>	
HARMEAN ↗	1868
$n \left(\sum_{i=1}^n x_i^{-1} \right)^{-1}$ <p>Returns the harmonic mean of a list of numeric values if their spread is not too great. Returns zero otherwise.</p>	

MEAN

Returns the arithmetic mean of a list of numeric values.



Calculates the arithmetic mean of non-missing values:

$$\frac{1}{n} \sum_{i=1}^n x_i$$

where n is the number of non-missing values in the list.

Return type: Numeric

value**Type:** Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the arithmetic mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  m1 = MEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "");
  PUT m1=;
  m2 = MEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 1e15);
  PUT m2=;
RUN;
```

This produces the following output:

```
m1=1.8333333333
m2=10e14
```

The calculation is as follows:

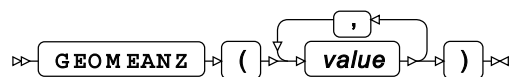
$$m_1 = \frac{1}{9} (1.1+1.2+1.3+1.4+1.5+1+2+3+4) = 1.8333333333$$

$$m_2 = \frac{1}{10} (1.1+1.2+1.3+1.4+1.5+1+2+3+4+10^{15}) = 10^{14}$$

The outlier value in the second example results in a much greater mean.

GEOMEANZ

Returns the geometric mean of a list of numeric values.



Calculates the geometric mean of non-missing values:

$$\left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}$$

where n is the number of non-missing values in the list.

Note:

If one of the values is zero, the result is also always zero.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Restriction: $value \geq 0$

If the argument is out of range, a missing value is returned.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the geometric mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  m1 = GEOMEANZ(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "");
  PUT m1=;
  m2 = GEOMEANZ(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 1e15);
  PUT m2=;
RUN;
```

This produces the following output:

```
m1=1.6414075414
m2=49.396341221
```

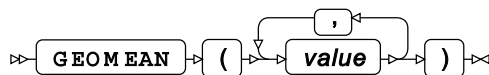
The calculation is as follows:

$$m_1 = (1.1 \cdot 1.2 \cdot 1.3 \cdot 1.4 \cdot 1.5 \cdot 1 \cdot 2 \cdot 3 \cdot 4)^{\frac{1}{9}} = 1.6414075414$$

$$m_2 = (1.1 \cdot 1.2 \cdot 1.3 \cdot 1.4 \cdot 1.5 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdot 10^{15})^{\frac{1}{10}} = 49.396341221$$

GEOMEAN

Returns the geometric mean of a list of numeric values if their spread is not too great. Returns zero otherwise.



Calculates the geometric mean of non-missing values:

$$\left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}$$

where n is the number of non-missing values in the list.

Note:

If one of the values is zero, the result is also always zero.

Function `GEOMEAN` uses the same formula as function `GEOMEANZ` (see [GEOMEANZ](#) (page 1864)), but if the difference between the minimum and maximum values (the spread) exceeds 10^{13} , zero is returned instead of the actual mean.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Restriction: $value \geq 0$

If the argument is out of range, a missing value is returned.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the geometric mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  m1 = GEOMEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "");
  PUT m1=;
  m2 = GEOMEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 1e15);
  PUT m2=;
RUN;
```

This produces the following output:

```
m1=1.6414075414
m2=0
```

The calculation is as follows:

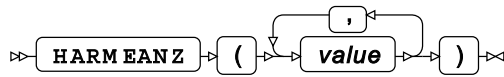
$$m_1 = \left\{ \begin{array}{l} \text{spread is } 4 - 1 < 10^{13} \\ \text{then return geometric mean} \\ (1.1 \cdot 1.2 \cdot 1.3 \cdot 1.4 \cdot 1.5 \cdot 1 \cdot 2 \cdot 3 \cdot 4)^{\frac{1}{9}} \end{array} \right\} = 1.6414075414$$

$$m_2 = \left\{ \begin{array}{l} \text{spread is } 10^{15} - 1 > 10^{13} \\ \text{then return zero} \end{array} \right\} = 0$$

The output for m_1 is the same as that returned by the `GEOMEANZ` function. However, the output for m_2 is zero because the spread of the second value list is too great.

HARMEANZ

Returns the harmonic mean of a list of numeric values.



Calculates the harmonic mean of non-missing values:

$$n \left(\sum_{i=1}^n x_i^{-1} \right)^{-1}$$

where n is the number of non-missing values in the list.

If any of the values is zero, computation is not attempted and zero is returned.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Restriction: $value \geq 0$

If the argument is out of range, a missing value is returned.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the harmonic mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  m1 = HARMEANZ(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "");
  PUT m1=;
  m2 = HARMEANZ(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 1e15);
  PUT m2=;
  m3 = HARMEANZ(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 0);
  PUT m3=;
RUN;
```

This produces the following output:

```
m1=1.5060390343
m2=1.6733767048
m3=0
```

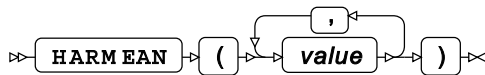
The calculation is as follows:

$$\begin{aligned}
 m_1 &= 9 \left(\frac{1}{1.1} + \frac{1}{1.2} + \frac{1}{1.3} + \frac{1}{1.4} + \frac{1}{1.5} + \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \right)^{-1} = 1.5060390343 \\
 m_2 &= 10 \left(\frac{1}{1.1} + \frac{1}{1.2} + \frac{1}{1.3} + \frac{1}{1.4} + \frac{1}{1.5} + \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{10^{15}} \right)^{-1} = 1.6733767048 \\
 m_3 &= 0
 \end{aligned}$$

The third example contains a zero value in the list. Attempting to compute harmonic mean would result in division by zero, therefore computation is not attempted and zero is returned.

HARMEAN

Returns the harmonic mean of a list of numeric values if their spread is not too great. Returns zero otherwise.



Calculates the harmonic mean of non-missing values:

$$n \left(\sum_{i=1}^n x_i^{-1} \right)^{-1}$$

where n is the number of non-missing values in the list.

If any of the values is zero, computation is not attempted and zero is returned.

Function `HARMEAN` uses the same formula as function `HARMEANZ` (see [HARMEANZ](#) (page 1867)), but if the difference between the minimum and maximum values (the spread) exceeds 10^{13} , zero is returned instead of the actual mean.

Return type: Numeric

value

Type: Numeric

The value to be evaluated.

Restriction: $value \geq 0$

If the argument is out of range, a missing value is returned.

If all specified values are missing values, a missing value is returned.

Examples

In these examples, the harmonic mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  m1 = HARMEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "");
  PUT m1=;
  m2 = HARMEAN(1.1,1.2,1.3,1.4,1.5,1,2,3,., "a", "4", "", 1e15);
  PUT m2=;
RUN;
```

This produces the following output:

```
m1=1.5060390343
m2=0
```

The calculation is as follows:

$$m_1 = \begin{cases} \text{spread is } 4 - 1 < 10^{13} \\ \text{then return harmonic mean} \end{cases} = 1.5060390343$$

$$m_2 = \begin{cases} \text{spread is } 10^{15} - 1 > 10^{13} \\ \text{then return zero} \end{cases} = 0$$

The output for m_1 is the same as that returned by the `HARMEANZ` function. However, the output for m_2 is zero because the spread of the second value list is too great.

Variance, skewness and kurtosis calculations

Calculate functions based on the moments about the mean: variance, skewness, kurtosis and related.

Moment	Derived statistics	Related values
Second moment about the mean	Variance	Coefficient of variance, standard deviation and standard error
$m_2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$		
Third moment about the mean	Skewness	
$m_3 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^n (x_i - \bar{x})^3$		
Fourth moment about the mean	Kurtosis	
$m_4 = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n (x_i - \bar{x})^4 - \frac{3}{(n-2)(n-3)} \left(\sum_{i=1}^n (x_i - \bar{x})^2 \right)^2$		

The centre of the moments is the arithmetic mean (see [MEAN](#) (page 1863)):

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

The formulas in this section define statistics that estimate the corresponding population parameters. A *population parameter* is a characteristic based on the entire population of values, whereas a *statistic* is an estimate of a population parameter based on a sample of values randomly drawn from the population. In general, unless the complete population is known and available for computation, it is not possible to calculate a population parameter directly, so statistics are used instead. The same population parameter can have several statistics associated with it, as it can be estimated in several ways.

Population parameters and some statistics often have widely known identifiers, such as the ones used in this section. Such identifiers may not be adopted universally, and may sometimes be used to denote other entities. Please confirm your notation if unsure.

For more information on population parameters, statistics, moments about the mean as well as identifiers as used in this section, see for example D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Second Edition, Boca Raton, Florida: Chapman & Hall/CRC (2000).

Examples in this section use two similar lists of values. The first list contains values close to each other, while the second list contains the same values plus a significantly larger outlier:

1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", ""

1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15

VAR [↗](#)..... 1871

$$\tilde{s}^2 = m_2$$

Returns the variance of a list of numeric values.

CV [↗](#)..... 1872

$$c_v = \frac{\sqrt{\tilde{s}^2}}{\bar{x}} \cdot 100$$

Returns the coefficient of variation of a list of numeric values.

STD [↗](#)..... 1873

$$\tilde{s} = \sqrt{\tilde{s}^2}$$

Returns the standard deviation of a list of numeric values.

STDERR [↗](#)..... 1874

$$SE = \frac{\tilde{s}}{\sqrt{n}}$$

Returns the standard error of the mean of a list of numeric values.

SKEWNESS [↗](#)..... 1875

$$g_1 = \frac{m_3}{\tilde{s}^3} = \frac{1}{\tilde{s}^3} \cdot \frac{n}{(n-1)(n-2)} \sum_{i=1}^n (x_i - \bar{x})^3$$

Returns the skewness of a list of numeric values.

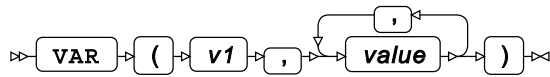
KURTOSIS [↗](#)..... 1878

$$g_2 = \frac{m_4}{\tilde{s}^4} = \frac{1}{\tilde{s}^4} \cdot \left[\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n (x_i - \bar{x})^4 - \frac{3}{(n-2)(n-3)} \left(\sum_{i=1}^n (x_i - \bar{x})^2 \right)^2 \right]$$

Returns the kurtosis of a list of numeric values.

VAR

Returns the variance of a list of numeric values.



Requires at least two arguments. Calculates the variance \tilde{s}^2 of non-missing values, or the second moment about the mean. The returned statistic is the unbiased estimate of the population variance σ^2 . See *Variance, skewness and kurtosis calculations* [↗](#) (page 1869) for more information on moments about the mean, population parameters and statistics.

$$\tilde{s}^2 = m_2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

where n is the number of non-missing values in the list.

Return type: Numeric

If only one value is specified or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Examples

In these examples, the variance of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = VAR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT s1=;
  s2 = VAR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT s2=;
RUN;
```

This produces the following output:

```
s1=2.6486111111
s2=1e29
```

The calculation is as follows:

$$\begin{aligned}\bar{x}_1 &= \text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) = 1.3888888889 \approx 1.39 \\ \bar{x}_2 &= \text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) = 10^{14}\end{aligned}$$

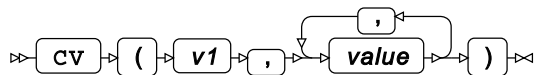
$$\tilde{s}_1^2 = \frac{1}{9-1} \cdot \left[\begin{aligned} &(1.1-1.39)^2 + (1.2-1.39)^2 + (1.3-1.39)^2 \\ &+ (1.4-1.39)^2 + (1.5-1.39)^2 + (1-1.39)^2 \\ &+ (-2-1.39)^2 + (3-1.39)^2 + (4-1.39)^2 \end{aligned} \right] = 2.6486111111$$

$$\tilde{s}_2^2 = \frac{1}{10-1} \cdot \left[\begin{aligned} &(1.1-10^{14})^2 + (1.2-10^{14})^2 + (1.3-10^{14})^2 \\ &+ (1.4-10^{14})^2 + (1.5-10^{14})^2 + (1-10^{14})^2 \\ &+ (-2-10^{14})^2 + (3-10^{14})^2 + (4-10^{14})^2 + (10^{15}-10^{14})^2 \end{aligned} \right] = 10^{29}$$

The outlier in the second sample dramatically increases the variance.

CV

Returns the coefficient of variation of a list of numeric values.



Requires at least two arguments. Calculates the coefficient of variation c_v of non-missing values expressed in percent, using their the variance \tilde{s}^2 (see [VAR](#) (page 1871)) and their the arithmetic mean \bar{x} (see [MEAN](#) (page 1863)):

$$c_v = \frac{\sqrt{\tilde{s}^2}}{\bar{x}} \cdot 100 = \frac{\sqrt{\text{VAR}(x_1, \dots, x_n)}}{\text{MEAN}(x_1, \dots, x_n)} \cdot 100$$

where n is the number of non-missing values in the list.

Return type: Numeric

If only one value is specified or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Examples

In these examples, the coefficient of variation of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  c1 = CV(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT c1=;
  c2 = CV(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT c2=;
RUN;
```

This produces the following output:

```
c1=117.17678951
c2=316.22776602
```

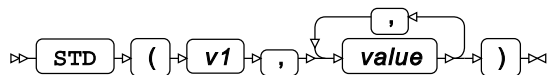
The calculation is as follows:

$$c_1 = \frac{\sqrt{\text{VAR}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, .)}}{\text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, .)} \cdot 100 = \frac{\sqrt{2.6486111111}}{1.3888888889} \cdot 100 = 117.17678951$$

$$c_2 = \frac{\sqrt{\text{VAR}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 10^{15})}}{\text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 10^{15})} \cdot 100 = \frac{\sqrt{10^{29}}}{10^{14}} \cdot 100 = 316.22776602$$

STD

Returns the standard deviation of a list of numeric values.



Requires at least two arguments. Calculates the standard deviation \tilde{s} of non-missing values using their the variance \tilde{s}^2 , see [VAR](#) (page 1871). The returned statistic is the unbiased estimate of the population standard deviation σ . See [Variance, skewness and kurtosis calculations](#) (page 1869) for more information on moments about the mean, population parameters and statistics.

$$\tilde{s} = \sqrt{\tilde{s}^2} = \sqrt{\text{VAR}(x_1, \dots, x_n)}$$

where n is the number of non-missing values in the list.

Return type: Numeric

If only one value is specified or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Examples

In these examples, the standard deviation of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  s1 = STD(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT s1=;
  s2 = STD(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT s2=;
RUN;
```

This produces the following output:

```
s1=1.6274554099
s2=3.1622777e14
```

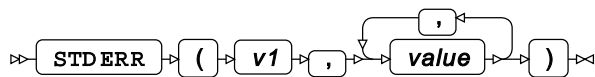
The calculation is as follows:

$$s_1 = \sqrt{\text{VAR}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4)} = \sqrt{2.6486111111} = 1.6274554099$$

$$s_2 = \sqrt{\text{VAR}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15})} = \sqrt{10^{29}} = 3.1622777 \cdot 10^{14}$$

STDERR

Returns the standard error of the mean of a list of numeric values.



Requires at least two arguments. Calculates the standard error of the mean SE of non-missing values using their the standard deviation \tilde{s} , see [STD](#) (page 1873):

$$SE = \frac{\tilde{s}}{\sqrt{n}} = \frac{\text{STD}(x_1, \dots, x_n)}{\sqrt{n}}$$

where n is the number of non-missing values in the list.

Return type: Numeric

If only one value is specified or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

value

Type: Numeric

Further value to be evaluated.

Examples

In these examples, the standard error of the mean of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  se1 = STDERR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "");
  PUT se1=;
  se2 = STDERR(1.1,1.2,1.3,1.4,1.5,1,-2,3,., "a", "4", "", 1e15);
  PUT se2=;
RUN;
```

This produces the following output:

```
se1=0.5424851366
se2=1e14
```

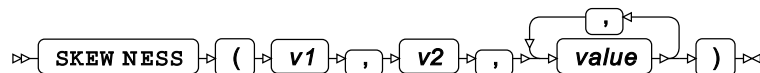
The calculation is as follows:

$$se_1 = \frac{STD(1.1,1.2,1.3,1.4,1.5,1,-2,3)}{\sqrt{9}} = \frac{1.6274554099}{\sqrt{9}} = 0.5424851366$$

$$se_2 = \frac{STD(1.1,1.2,1.3,1.4,1.5,1,-2,3,10^{15})}{\sqrt{10}} = \frac{3.1622777 \cdot 10^{14}}{\sqrt{10}} = 10^{14}$$

SKEWNESS

Returns the skewness of a list of numeric values.



Requires at least three arguments. Calculates the skewness g_1 of non-missing values as a ratio of the third moment about the mean m_3 and cubed standard deviation \tilde{s}^3 , see [STD](#) (page 1873).

The returned statistic is the unitless estimate of the population parameter γ_1 . See [Variance, skewness and kurtosis calculations](#) (page 1869) for more information on moments about the mean, population parameters and statistics.

$$g_1 = \frac{m_3}{\tilde{s}^3} = \frac{1}{\tilde{s}^3} \cdot \frac{n}{(n-1)(n-2)} \sum_{i=1}^n (x_i - \bar{x})^3$$

where n is the number of non-missing values in the list.

Return type: Numeric

If fewer than three values are available or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

v2

Type: Numeric

The second value in the list.

value

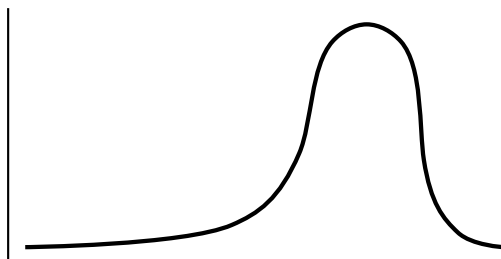
Type: Numeric

Further value to be evaluated.

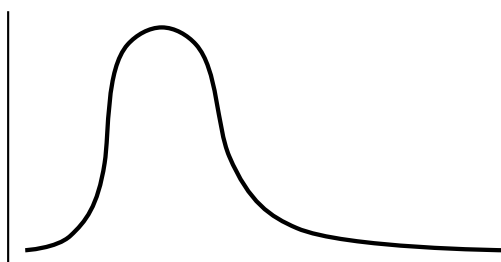
Skewness is a measure of symmetry of a distribution as illustrated below:



Symmetrical distribution $g1 = 0$



Negatively skewed distribution $g1 < 0$



Positively skewed distribution $g1 > 0$

Examples

In these examples, the skewness of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
g1_1 = SKEWNESS(1.1,1.2,1.3,1.4, 1.5, 1, -2, 3, ., "a", "4", "");
PUT g1_1=;
g1_2 = SKEWNESS(1.1,1.2,1.3,1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15);
PUT g1_2=;
g1_3 = SKEWNESS(1.1,1.2,1.3,1.4, 1.5, 1, -2, 3, ., "a", "4", "", -1e15);
PUT g1_3=;
g1_4 = SKEWNESS(1.1,1.2,1.3,1.4,-1.1,-1.2,-1.3,-1.4, ., "a", "0", "");
PUT g1_4=;
RUN;
```

This produces the following output:

```
g1_1=-0.634755445
g1_2=3.1622776602
g1_3=-3.1622776602
g1_4=-9.88457e-17
```

The calculation is as follows:

$$\begin{aligned}\bar{x}_1 &= \text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) = 1.3888888889 \approx 1.39 \\ \bar{x}_2 &= \text{MEAN}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) = 10^{14} \\ \tilde{s}_1 &= \text{STD}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) = 1.6274554099 \approx 1.6 \\ \tilde{s}_2 &= \text{STD}(1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) = 3.1622777 \cdot 10^{14} \approx 3 \cdot 10^{14} \\ g_{1_1} &= \frac{1}{1.6^3} \cdot \frac{9}{(9-1)(9-2)} \cdot \left[\frac{(1.1-1.39)^3 + (1.2-1.39)^3 + (1.3-1.39)^3}{+ (1.4-1.39)^3 + (1.5-1.39)^3 + (1-1.39)^3} + \frac{(-2-1.39)^3 + (3-1.39)^3 + (4-1.39)^3}{+ (10^{15}-1.39)^3} \right] = -0.634755445 \\ g_{1_2} &= \frac{1}{(3 \cdot 10^{14})^3} \cdot \frac{10}{(10-1)(10-2)} \cdot \left[\frac{(1.1-10^{14})^3 + (1.2-10^{14})^3 + (1.3-10^{14})^3}{+ (1.4-10^{14})^3 + (1.5-10^{14})^3 + (1-10^{14})^3} + \frac{(-2-10^{14})^3 + (3-10^{14})^3 + (4-10^{14})^3}{+ (10^{15}-10^{14})^3} \right] = 3.1622776602\end{aligned}$$

Similarly to the above, $g_{1_3} = -3.1622776602$ and $g_{1_4} = -9.88457 \cdot 10^{-17} \approx 0$.

The positive outlier in the second sample significantly increases the degree of skewness and changes its sign, as the sample is now skewed to the right. Equally large negative outlier in the third sample yields equally large skewness, changing its direction to the left.

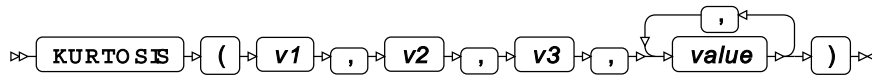
The fourth sample is symmetrical about the mean, and its skewness is close to zero.

Note:

Even for a perfectly symmetrical sample, the return value may not be exactly zero.

KURTOSIS

Returns the kurtosis of a list of numeric values.



Requires at least four arguments. Calculates the kurtosis g_2 of non-missing values as a ratio of the fourth moment about the mean m_4 and squared variance \tilde{s}^4 . The returned statistic is the unitless estimate of the population parameter γ_2 . See *Variance, skewness and kurtosis calculations* [↗](#) (page 1869) for more information on moments about the mean, population parameters and statistics.

$$g_2 = \frac{m_4}{\tilde{s}^4} = \frac{1}{\tilde{s}^4} \cdot \left[\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n (x_i - \bar{x})^4 - \frac{3}{(n-2)(n-3)} \left(\sum_{i=1}^n (x_i - \bar{x})^2 \right)^2 \right]$$

where n is the number of non-missing values in the list.

Return type: Numeric

If fewer than four values are available or if all specified values are missing values, a missing value is returned.

v1

Type: Numeric

The first value in the list.

v2

Type: Numeric

The second value in the list.

v3

Type: Numeric

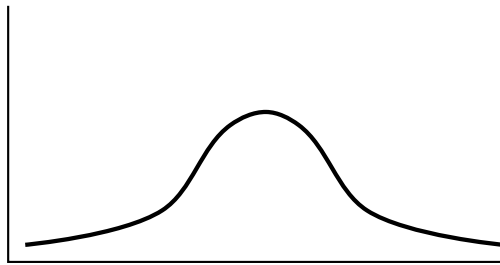
The third value in the list.

value

Type: Numeric

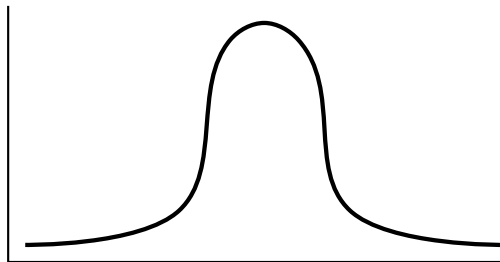
Further value to be evaluated.

Kurtosis characterises the degree of curvature or tail weight compared to a normal distribution as illustrated below:



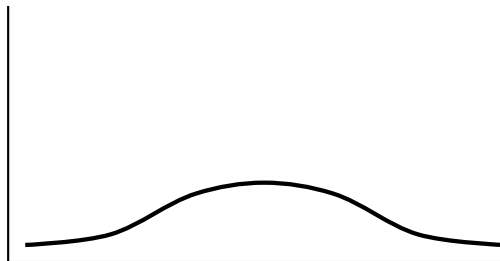
Mesokurtic distribution $g_2 = 0$

Normal distributions are mesokurtic, this is the reference distribution for kurtosis.



Leptokurtic distribution $g_2 > 0$

The values are clustered closer around the mean than in a normal distribution resulting in a higher peak and heavier tails.



Platykurtic distribution $g_2 < 0$

The values are spread further from the mean than in a normal distribution resulting in a lower peak and lighter tails.

Examples

In these examples, the kurtosis of the arguments is returned. The results are written to the log.

```
DATA _NULL_;
  g2_1 = KURTOSIS(1.1,1.2,1.3,1.4, 1.5, 1, -2, 3, ., "a", "4", "");
  PUT g2_1=;
  g2_2 = KURTOSIS(1.1,1.2,1.3,1.4, 1.5, 1, -2, 3, ., "a", "4", "", 1e15);
  PUT g2_2=;
  g2_3 = KURTOSIS(1.1,1.2,1.3,1.4, 1.1, 1.1, 1, 1, ., "a", "1", "");
  PUT g2_3=;
  g2_4 = KURTOSIS(1.1,1.2,1.3,1.4,-1.1,-1.2,-1.3,-1.4, ., "a", "0", "");
  PUT g2_4=;
RUN;
```

This produces the following output:

```
g2_1=2.4980234276
g2_2=10
g2_3=-0.017857143
g2_4=-2.360997732
```

The calculation is as follows:

$$\begin{aligned}\bar{x}_1 &= \text{MEAN} (1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) &= 1.3888888889 &\approx 1.39 \\ \bar{x}_2 &= \text{MEAN} (1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) &= 10^{14} \\ \tilde{s}_1 &= \text{STD} (1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4) &= 1.6274554099 &\approx 1.6 \\ \tilde{s}_2 &= \text{STD} (1.1, 1.2, 1.3, 1.4, 1.5, 1, -2, 3, 4, 10^{15}) &= 3.1622777 \cdot 10^{14} &\approx 3 \cdot 10^{14}\end{aligned}$$

$$g_{2_1} = \frac{1}{1.6^4} \cdot \frac{9(9+1)}{(9-1)(9-2)(9-3)} \cdot \left[\begin{aligned} &(1.1-1.39)^4 + (1.2-1.39)^4 + (1.3-1.39)^4 \\ &+ (1.4-1.39)^4 + (1.5-1.39)^4 + (1-1.39)^4 \\ &+ (-2-1.39)^4 + (3-1.39)^4 + (4-1.39)^4 \end{aligned} \right] \\ - \frac{1}{1.6^4} \cdot \frac{3}{(9-2)(9-3)} \cdot \left[\begin{aligned} &(1.1-1.39)^2 + (1.2-1.39)^2 + (1.3-1.39)^2 \\ &+ (1.4-1.39)^2 + (1.5-1.39)^2 + (1-1.39)^2 \\ &+ (-2-1.39)^2 + (3-1.39)^2 + (4-1.39)^2 \end{aligned} \right]^2 = 2.4980234276$$

$$g_{2_2} = \frac{1}{1.6^4} \cdot \frac{10(10+1)}{(10-1)(10-2)(10-3)} \cdot \left[\begin{aligned} &(1.1-1.39)^4 + (1.2-1.39)^4 + (1.3-1.39)^4 \\ &+ (1.4-1.39)^4 + (1.5-1.39)^4 + (1-1.39)^4 \\ &+ (-2-1.39)^4 + (3-1.39)^4 + (4-1.39)^4 \\ &+ (10^{15}-10^{14})^4 \end{aligned} \right] \\ - \frac{1}{1.6^4} \cdot \frac{3}{(10-2)(10-3)} \cdot \left[\begin{aligned} &(1.1-1.39)^2 + (1.2-1.39)^2 + (1.3-1.39)^2 \\ &+ (1.4-1.39)^2 + (1.5-1.39)^2 + (1-1.39)^2 \\ &+ (-2-1.39)^2 + (3-1.39)^2 + (4-1.39)^2 \\ &+ (10^{15}-10^{14})^2 \end{aligned} \right]^2 = 10$$

Analogous to the above, $g_{2_3} = -0.017857143 \approx 0$ and $g_{2_4} = -2.360997732$.

The first sample represents a distribution with heavier tails than a normal distribution, and the outlier in the second sample dramatically increases it.

The third sample has kurtosis close to zero making its tail weight similar to that of a normal distribution.

The fourth sample has negative kurtosis indicating that it has lighter tails than a normal distribution.

Memory manipulation functions

Manipulate memory.

You can find the addresses of memory locations for variables, and peek into and poke those locations. The functions are interrelated. For example, you can use `ADDRLONG` to get the memory address of a variable, and then use `PEEKLONG` to view the contents of that address, or `POKELONG` to write data to that address.

Note:

As these functions involve addressing memory locations, you should be careful when using them.

The group of functions with names ending in `LONG` can be used in both 64-bit and 32-bit environments. The functions whose names do *not* end in `LONG` are provided for compatibility with programs already written for 32-bit environments.

`ADDRLONG` [↗](#)..... 1881
Returns the address of a variable (64-bit only).

ADDRLONGX ↗	1882
Returns the address of a variable (64-bit only).	
PEEKCLONG ↗	1883
Returns the contents of a memory address (64-bit only), where the contents must be a string of one or more character	
PEEKLONG ↗	1884
Returns the contents of a number of bytes from a memory address (64-bit only), where the contents must be a number.	
PTRLONGADD ↗	1885
Returns the pointer address in a character variable.	
CALL POKELONG ↗	1886
Writes a specified sequence of bytes to a variable (64-bit only), at a specified memory location.	
Memory manipulation functions (32-bit compatibility) ↗	1888
Manipulate memory on 32-bit systems.	

ADDRLONG

Returns the address of a variable (64-bit only).

➤ `ADDRLONG` ➤ `(variable)` ➤

Return type: Character

The address location. To display the return formatting is required.

variable

Type: Var

The variable.

Example

In this example, the address of a variable in memory is returned. The result is written to the log.

```
DATA _NULL_;

FORMAT addr $HEX16.;
varval="https://www.worldprogramming.com/";
addr=ADDRLONG(VARVAL);

PUT varval=;
PUT addr=;

RUN;
```

This produces the following output:

```
varval=https://www.worldprogramming.com/  
addr=E06A6D3659020000
```

ADDRLONGX

Returns the address of a variable (64-bit only).

⇒ **ADDRLONGX** ⇒ (*variable-name*) ⇒

Return type: Character

The address location. To display the return formatting is required.

variable-name

Type: Character

The name of the variable.

When a function-name is append by 'X', variable names should be entered enclosed by quotes. However, it is also possible to assign the *variable-name*, for example, in the data steps below `abc = "varval"`. Variable `abc` (without the quotes) can therefore be entered instead.

Example

In this example, the address of variables in memory are returned. The result is written to the log.

```
DATA _NULL_;  
  
  FORMAT addr addr1 addr2 $HEX16.;  
  varval="https://www.worldprogramming.com/";  
  addr=ADDRLONGX("varval");  
  
  abc = "varval";  
  addr1=ADDRLONGX(abc);  
  addr2=ADDRLONGX("abc");  
  
  PUT varval=;  
  PUT abc=;  
  
  PUT addr=;  
  PUT addr1=;  
  PUT addr2=;  
  
RUN;
```


This produces the following output:

```
varval=https://www.worldprogramming.com/  
abc=varval  
addr=FC66080120202020  
addr1=FC66080120202020  
addr2=3567080120202020
```

In this example, when the `abc` variable is enclosed in quotes, it is interpreted as a different string variable and not as an assigned variable name, hence a new address is outputted in `addr2`.

PEEKCLONG

Returns the contents of a memory address (64-bit only), where the contents must be a string of one or more character



Return type: Character

address

Type: Character

The address of the string in memory.

length

Optional argument

Type: Numeric

Starting from the first character, the number of characters that you want returned. If no value is applied, the full string is returned as a default.

This function expects to process characters, any numerical entry will not be processed. If you want to process numbers then use the `PEEKLONG` function.

Example

In this example, the contents of the address are truncated at character 43 when the `addr` is returned. The result is written to the log.

```
DATA _NULL_;  
FORMAT  
  addr $HEX16.  
  myvar value $70.;  
myvar = "There was a very young man from California, who had to visit Oklahoma.";  
addr = ADDRLONG(myvar);  
value = PEEKCLONG(addr, 43);  
PUT myvar=;  
PUT addr=;  
PUT value=;  
RUN;
```

This produces the following output:

```
myvar=There was a very young man from California, who had to visit Oklahoma.  
addr=006BE26B72010000  
value=There was a very young man from California,
```

PEEKLONG

Returns the contents of a number of bytes from a memory address (64-bit only), where the contents must be a number.



Return type: Numeric

The memory address contents. To display the return formatting is required.

address

Type: Character

The address of the number in memory.

length

Optional argument

Type: Numeric

The number of bytes to be returned.

Valid values for length are 1, 2, 3, 4, and 8.

The default value of length is 4.

Lengths of 1, 2, and 4 return 8, 16, and 32-bit integers respectively.

A length of 3 returns a 32-bit integer formed from the first three bytes of the source address.

A length of 8 returns a double.

This function expects to process a number, any other entry will generate an error. If you want to process characters then use the `PEEKCLONG` function.

Example

In this example, the contents of two memory addresses are obtained. The result is written to the log.

```
DATA _NULL_;  
  FORMAT addr $HEX16.;  
  FORMAT number 8.;  
  
  var = 36;  
  
  ARRAY NUMS(8) (1 2 3 4 5 6 7 8);  
  
  addr    = ADDRLONG(var);  
  number = PEEKLONG(addr, 8);  
  
  PUT addr= number=;  
  
  addr    = ADDRLONG(nums8);  
  number = PEEKLONG(addr, 8);  
  
  PUT addr= number=;  
  
RUN;
```

This produces the following output:

```
addr=F88BA10020202020 number=36  
addr=E046A10020202020 number=8
```

PTRLONGADD

Returns the pointer address in a character variable.

➤ **PTRLONGADD** ➤ (*address* , *amount*) ➤

Return type: Character

The pointer address. To display the return formatting is required.

address

Type: Character

The memory location.

amount**Type:** Numeric

The number of bytes to add to an address.

Example

In this example, the pointer address, memory address and the contents of the memory address in the variable `x` are returned. The result is written to the log.

Note:

The memory address is not a fixed value, and will change depending on the processing tasks of the computer.

```
DATA _NULL_;

x='12345678910 - very long';
ptr_add=ADDRLONG(x);

y=PTRLONGADD(ptr_add,0);

mem_add=PEEKLONG(y);
con_mem=PEEKCLONG(y);

PUT y HEX16. " = " "Pointer address in HEX";
PUT " ";
PUT mem_add "= " "Memory address";
PUT " ";
PUT con_mem "= " "Contents of the memory address in the variable x";
PUT " ";

RUN;
```

This produces the following output:

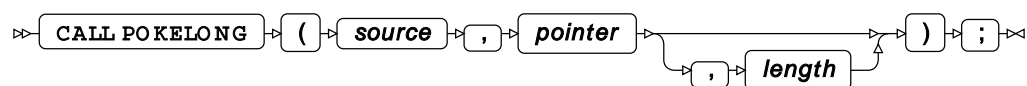
```
386B6B367F010000 = Pointer address in HEX

875660417 = Memory address

12345678 = Contents of the memory address in the variable x
```

CALL POKELONG

Writes a specified sequence of bytes to a variable (64-bit only), at a specified memory location.



source**Type:** Character

The sequence of bytes.

pointer**Type:** Character

The memory location to write to.

length

Optional argument

Type: Numeric

The length of the source (in bytes) to write. The default is the full length of the string.

Example 1 – using the default length

In this example, the `source` string is copied to the contents of the destination string variable. The result is written to the `log`.

```
DATA _NULL_;

scr = "https://www.worldprogramming.com/ ";

dest = "***** --- Test ---";

ptr=ADDRLONG(dest);
CALL POKELONG(scr, ptr);
PUT "Result: " dest;

PUT "      ";
RUN;
```

This produces the following output:

```
Result: https://www.worldprogramming.com/ --- Test ---
```

Example 2 – using a specified length

In this example, 17 characters of the source string are copied to the destination string variable. The result is written to the log.

```
DATA _NULL_;

scr = "https://www.worldprogramming.com/ ";

dest = "***** --- Test ---";

ptr=ADDRLONG(dest);
CALL POKELONG(scr, ptr, 17);
PUT "Result: " dest;

PUT " ";
RUN;
```

This produces the following output:

```
Result: https://www.world***** --- Test ---
```

Memory manipulation functions (32-bit compatibility)

Manipulate memory on 32-bit systems.

These functions are provided for compatibility with existing programs.

ADDR

Returns the address of a variable (32-bit only).

➤ **ADDR** ➤ **(variable)** ➤

Return type: Numeric

The address location. To display the return formatting is required.

variable

Type: Var

The variable.

Example

In this example, the address of a variable in memory is returned. The result is written to the log.

```
DATA _NULL_;  
  
FORMAT addr $HEX16.;  
varval="https://www.worldprogramming.com/";  
addr=ADDR(varval);  
  
PUT varval=;  
PUT addr=;  
  
RUN;
```

This produces the following output:

```
varval=https://www.worldprogramming.com/  
addr=2039383536373234
```

ADDRX

Returns the address of a variable (32-bit only).

➤ **ADDRX** ➤ (*variable-name*) ➤

Return type: Numeric

The address location. To display the return formatting is required.

variable-name

Type: Character

The name of the variable.

When a function-name is append by 'X', variable names should be entered enclosed by quotes. However, it is also possible to assign the *variable-name*, for example, in the data steps below `abc = "varval"`. Variable `abc` (without the quotes) can therefore be entered instead.

Example

In this example, the address of variables in memory are returned. The result is written to the log.

```
DATA _NULL_;

FORMAT addr addr1 addr2 $HEX16.;
VARVAL="https://www.worldprogramming.com/";
addr=ADDRX("varval");

abc = "varval";
addr1=ADDRX(abc);
addr2=ADDRX("abc");

PUT varval=;
PUT abc=;

PUT addr=;
PUT addr1=;
PUT addr2=;

RUN;
```

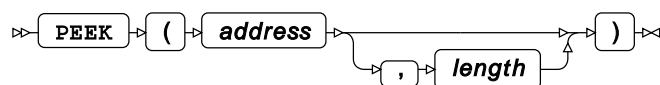
This produces the following output:

```
varval=https://www.worldprogramming.com/
abc=varval
addr=3636393338363434
addr1=3636393338363434
addr2=3636393338363737
```

In this example, when the `abc` variable is enclosed in quotes, it is interpreted as a different string variable and not as an assigned variable name, hence a new address is outputted in `addr2`.

PEEK

Returns the contents of a number of bytes from a memory address (32-bit only), where the contents must be a number.



Return type: Numeric

The memory address contents. To display the return formatting is required.

address

Type: Numeric

The address of the number in memory.

length

Optional argument

Type: Numeric

The number of bytes to be returned.

Valid values for length are 1, 2, 3, 4, and 8.

The default value of length is 4.

Lengths of 1, 2, and 4 return 8, 16, and 32-bit integers respectively.

A length of 3 returns a 32-bit integer formed from the first three bytes of the source address.

A length of 8 returns a double.

This function expects to process a number, any other entry will generate an error. If you want to process characters then use the PEEKC function.

Example

In this example, the contents of two memory addresses are obtained. The result is written to the log.

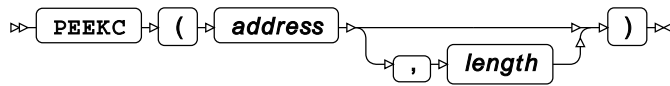
```
DATA _NULL_;  
  FORMAT addr $HEX16.;  
  FORMAT number 8.;  
  
  ARRAY NUMS(8) (1 2 3 4 5 6 7 8);  
  
  var = 54;  
  
  addr = ADDR(var);  
  number = PEEK(addr, 8);  
  
  PUT addr= number=;  
  
  addr = ADDR(nums6);  
  number = PEEK(addr, 8);  
  
  PUT addr= number=;  
  
RUN;
```

This produces the following output:

```
addr=3832393338383536 number=54  
addr=2039373137343536 number=6
```

PEEKC

Returns the contents of a memory address (32-bit only), where the contents must be a string of one or more characters.



Return type: Character

The contents of the memory address. To display the return formatting is required.

address

Type: Numeric

The address of the string in memory.

length

Optional argument

Type: Numeric

Starting from the first character, the number of characters that you want returned. If no value is applied, the full string is returned as a default.

This function expects to process characters, any numerical entry will not be processed. If you want to process numbers then use the `PEEK` function.

Example

In this example, the contents of the address are truncated at character 43 when the address is returned. The result is written to the log.

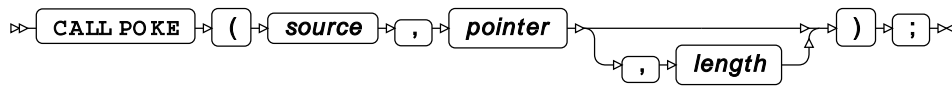
```
DATA _NULL_;  
  FORMAT  
    addr $HEX16.  
  myvar value $70.;  
  myvar = "There was a very young man from California, who had to visit Oklahoma.";  
  addr = ADDR(myvar);  
  value = PEEKC(addr, 43);  
  PUT myvar=;  
  PUT addr=;  
  PUT value=;  
RUN;
```

This produces the following output:

```
myvar=There was a very young man from California, who had to visit Oklahoma.  
addr=3132323136303336  
value=There was a very young man from California,
```

CALL POKE

Writes a specified sequence of bytes to a variable (32-bit only), at a specified memory location.



source

Type: Character

The sequence of bytes.

pointer

Type: Numeric

The memory location to write to.

length

Optional argument

Type: Numeric

The length of the source (in bytes) to write. The default is the full length of the string.

Example 1 – using the default length

In this example, the `source` string is copied to the contents of the destination string variable. The result is written to the `log`.

```
DATA _NULL_;

scr = "https://www.worldprogramming.com/ ";

dest = "***** --- Test ---";

ptr=ADDR(dest);
CALL POKE(scr, ptr);
PUT "Result: " dest;

PUT " ";
RUN;
```

This produces the following output:

```
Result: https://www.worldprogramming.com/ --- Test ---
```

Example 2 – using a specified length

In this example, 12 characters of the source string are copied to the destination string variable. The result is written to the log.

```
DATA _NULL_;

scr = "https://www.worldprogramming.com/ ";

dest = "***** --- Test ---";

ptr=ADDR(DEST);
CALL POKE(scr, ptr, 12);
PUT "Result: " dest;

PUT " ";
RUN;
```

This produces the following output:

```
Result: https://www.***** --- Test ---
```

Miscellaneous functions

Miscellaneous functions.

LOOKSLIKENUMBER ↗	1894
Returns a value to indicate whether a string might be a representation of a number.	
JSONPP ↗	1895
Returns formatted JSON code.	
SLEEP ↗	1897
Sets a time interval in which the system processing of instructions is suspended.	
CALL SLEEP ↗	1898
Sets a time interval during which the system processing of instructions is suspended.	
CALL SOUND ↗	1899
Creates a sound of a specified frequency and duration (Windows only).	

LOOKSLIKENUMBER

Returns a value to indicate whether a string might be a representation of a number.



Return type: Numeric

1 if positive, or 0 if not positive.

argument

Type: Character

The variable.

Example

In this example, variables of different values are used with the function. The result is written to the log.

```
DATA _NULL_;

  letters = "ABCDEF";
  sp = "1.175494351E-38";
  dp = "1.7976931348623158";
  paddle = "208.77E10";

  var = LOOKSLIKENUMBER (letters);
  IF var = '1' THEN PUT "The value of the letters variable looks like a number: "
var;
  ELSE PUT "The value of the letters variable is not a number: " var;

  var = LOOKSLIKENUMBER (sp);
  IF var = '1' THEN PUT "The value of the sp variable looks like a number: " var;
  ELSE PUT "The value of the sp variable is not a number: " var;

  var = LOOKSLIKENUMBER (dp);
  IF var = '1' THEN PUT "The value of the dp variable looks like a number: " var;
  ELSE PUT "The value of the dp variable is not a number: " var;

  var = LOOKSLIKENUMBER (paddle);
  IF var = '1' THEN PUT "The value of the paddle variable looks like a number: "
var;
  ELSE PUT "The value of the paddle variable is not a number: " var;

RUN;
```

This produces the following output:

```
The value of the letters variable is not a number: 0
The value of the sp variable looks like a number: 1
The value of the dp variable looks like a number: 1
The value of the paddle variable looks like a number: 1
```

JSONPP

Returns formatted JSON code.

⇒ **JSONPP** ⇒ (⇒ **input-file** ⇒ , ⇒ **output-file** ⇒) ⇒

An input file of unformatted or badly-formatted JSON code can be formatted.

Return type: Numeric

0	Successful
20004	The file does not exist, or does not contain valid JSON

input-file

Type: Character

The pathname and filename of the file to be formatted.

output-file

Type: Character

The pathname and filename of the formatted file.

The input file must contain valid JSON, otherwise an error is returned.

Note:

- Some applications, such as Notepad, cannot correctly display the line breaks of the formatted output; the line breaks can be displayed correctly in other applications, such as Notepad++.
- If the length of a line in an input file is set to a value that is shorter than the length of the lines in the input file (for example, by using the `LRECL` system option or the `LRECL` option on a `FILENAME` statement), the input file appears to contain invalid JSON and an error is returned.

Note:

Note:

Example

In this example, formatted output is created from an unformatted input file. The result is written to the log.

```
DATA _NULL_;

    rc = JSONPP("c:\temp\jsin","c:\temp\jsout");
    PUT rc;

RUN;
```

This formats the badly-formatted JSON code in the input file:

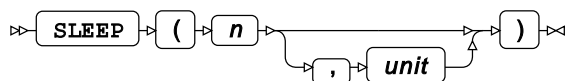
```
{ "menu" : { "id" : "file", "popup" : { "menuitem" : [{
    "onclick" : "CreateNewDoc()", "value" : "New"},
    {
    "onclick" : "OpenDoc()", "value" : "Open"},
    { "onclick" : "CloseDoc()", "value" : "Close"} ]},
    "value" : "File" }
```

and produces an output file containing the formatted JSON:

```
{
  "menu" : {
    "id" : "file",
    "popup" : {
      "menuitem" : [
        {
          "onclick" : "CreateNewDoc()",
          "value" : "New"
        },
        {
          "onclick" : "OpenDoc()",
          "value" : "Open"
        },
        {
          "onclick" : "CloseDoc()",
          "value" : "Close"
        }
      ]
    },
    "value" : "File"
  }
}
```

SLEEP

Sets a time interval in which the system processing of instructions is suspended.



Return type: Numeric

The suspended time in seconds.

n

Type: Numeric

If `unit` is specified, the number of units; otherwise the number of seconds in Windows, or the number of milliseconds in other operating systems, for example, Linux.

unit

Optional argument

Type: Numeric

The number of units, where a `unit` is one second.

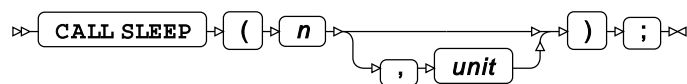
Example

In this example, the system processing is suspended for 15 seconds.

```
DATA _NULL;  
  
  number = 5;  
  unit = 3;  
  
  VAR = SLEEP (number, unit);  
RUN;
```

CALL SLEEP

Sets a time interval during which the system processing of instructions is suspended.



n

Type: Numeric

If `unit` is specified, the number of units; otherwise the number of milliseconds.

unit

Optional argument

Type: Numeric

The number of units, where a `unit` is one second.

Example 1

In this example, the system processing is suspended for five seconds.

```
DATA _NULL;  
  CALL SLEEP (5, 1);  
RUN;
```

Example 2

In this example, the system processing is suspended for 15 seconds.

```
DATA _NULL;  
  
  number = 5;  
  unit = 3;  
  
  CALL SLEEP (number, unit);  
RUN;
```


CALL SOUND

Creates a sound of a specified frequency and duration (Windows only).

➤ **CALL SOUND** ➤ (*frequency* , *duration*) ➤ ; ➤

frequency

Type: Numeric

The frequency of sound. This is specified in Hz in the range from 20 to 20000.

duration

Type: Numeric

The duration of sound. This is specified in seconds.

Example – common hearing range

In this example, the function uses a loop to output frequency ranges from 20Hz to 20,000Hz. This range is the common hearing range. The function then calls a very low frequency followed by a very high frequency. The use of headphones is advised.

```
DATA _NULL;

    frequency = 20;
    duration = 1000;

    DO i = 20 to 25;

        CALL SOUND (frequency, duration);

        frequency = frequency + 3330;

    END;

    CALL SOUND (60, 1000);
    CALL SOUND (1800, 1000);
RUN;
```

National language support functions

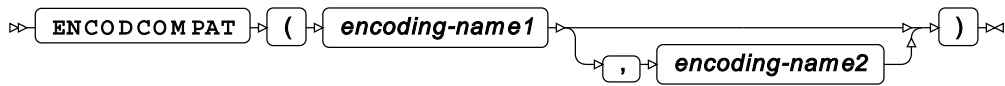
Acquire and set information relating to locales.

ENCODCOMPAT [🔗](#)..... 1900
Returns a value indicating whether an encoding name is compatible with the WPS session, or whether two encoding names are compatible with each other.

ENCODISVALID ↗	1902
Returns a value indicating whether an encoding name is valid and supported.	
GETLOCENV ↗	1903
Returns the abbreviation that specifies the type of character set of the current session locale.	
GETPXLANGUAGE ↗	1904
Returns a two-character code identifying the language setting for the locale.	
GETPXLOCALE ↗	1905
Either returns the locale code of the current server, or the locale code of a specified locale.	
GETPXREGION ↗	1906
Returns a two-character region code for a locale.	
NLDATE ↗	1907
Returns a date in a locale specific format.	
NLDATM ↗	1909
Returns a datetime in a locale specific format.	
NLTIME ↗	1912
Returns a time or datetime in a locale specific format for hour, minute, second, and AM/PM.	

ENCODCOMPAT

Returns a value indicating whether an encoding name is compatible with the WPS session, or whether two encoding names are compatible with each other.



Return type: Numeric

If you are checking compatibility with the current session, the return value can be:

Value	Description
1	The encoding is known and compatible.
0	The encoding is known but incompatible (transcoding will occur between the two encodings).

If you are checking compatibility between encodings, the return value can be:

Value	Description
2	The encodings are the same, apart from a newline character.
1	The encodings are known and compatible.
0	The encodings are known but incompatible (transcoding will occur between the two encodings).
-1	The first encoding is not a valid encoding name.
-2	The second encoding is not a valid encoding name.

encoding-name1

Type: Character

The name of the first encoding to be checked for compatibility with a server, or to be compared with *encoding-name2*.

encoding-name2

Optional argument

Type: Character

The name of the second encoding name to be compared with *encoding-name1*.

Example – an encoding check

In this example, an encoding for a character set is checked to see if it is compatible with the WPS session. The result is written to the log.

```
DATA _NULL_;

test = ENCODCOMPAT('wlatin1');
PUT test;

RUN;
```

This produces the following output:

```
0
```

Example – checking if two encodings are compatible

In this example, two encodings are checked for compatibility. The result is written to the log.

```
DATA _NULL_;

test = ENCODCOMPAT('thai', 'iso-8859-11');
PUT "The result of the comparison between the encodings thai and iso-8859-11 is: "
test;
PUT " ";

test = ENCODCOMPAT('shift-jis', 'utf-8');
PUT "The result of the comparison between the encodings shift-jis and utf-8 is: "
test;

RUN;
```

This produces the following output:

```
The result of the comparison between the encodings thai and iso-8859-11 is: 1
The result of the comparison between the encodings shift-jis and utf-8 is: 0
```

ENCODISVALID

Returns a value indicating whether an encoding name is valid and supported.

➤ `ENCODISVALID` ➤ `(encoding-name)` ➤

Return type: Numeric

The return value is one of the following:

Value	Description
3	The encoding name is an alias.
2	The long encoding name is valid.
1	The short encoding name is valid.
0	The encoding name is not valid.

encoding-name

Type: Character

The name of the encoding to be checked.

Example

In this example, four encoding names are checked to see if they are valid and supported. The encoding names can be long, short, or an alias. The result is written to the log.

```
DATA _NULL_;

  test = ENCODISVALID ('ibm-1026');

  PUT "The result of checking encoding name ibm-1026 is: " test;

  test = ENCODISVALID ('utf-32be');

  PUT "The result of checking encoding name utf-32be is: " test;

  test = ENCODISVALID ('E037');

  PUT "The result of checking encoding name utf-32be is: " test;

  test = ENCODISVALID ('UTF-9');

  PUT "The result of checking encoding name UTF-9 is:      " test;

RUN;
```

This produces the following output:

```
The result of checking encoding name ibm-1026 is: 3
The result of checking encoding name utf-32be is: 2
The result of checking encoding name utf-32be is: 1
The result of checking encoding name UFF-9 is: 0
```

GETLOCENV

Returns the abbreviation that specifies the type of character set of the current session locale.

⇒ **GETLOCENV** () ⇐

The character encoding sets used in computing are single byte, double byte, and multibyte.

Return type: Character

The abbreviation. It can be:

Abbreviation	Description
SBCS	Single Byte Character Set.
DBCS	Double Byte Character Set.
MBCS	Multibyte Character Set.

Example

In this example, the abbreviation for the current session encoding character set are returned. The result is written to the log.

```
DATA _NULL_;

    test = GETLOCENV ();
    PUT test;

RUN;
```

This produces the following output:

```
MBCS
```

The statements `x = GETOPTION("locale");` and `PUT x;` can be added to the DATA step to provide the locale code for language and country.

GETPXLANGUAGE

Returns a two-character code identifying the language setting for the locale.

➤ GETPXLANGUAGE ➤ () ➤

Return type: Character

Example – a locale language setting code check

In this example, the locale language setting code in the local area is returned. The result is written to the log.

```
DATA _NULL_;

    test = GETPXLANGUAGE ();
    PUT test;

RUN;
```

This produces the following output:

```
en
```

The statements `x = GETOPTION("locale");` and `PUT x;` can be added to the DATA step to provide the locale code for language and country.

Example – how to find the language code for a locale

In this example, the locale language code for Belarus is returned. For only this example, the locale value for Belarus has to be set in the system. Locale values are listed in this document. See [LOCALE Values](#) (page 37) for more information. The setting for Belarus is given as `Byelorussian_Belarus`. This is then added to the `OPTIONS LOCALE` statement. The result is written to the log.

```
DATA _NULL_;

    OPTIONS LOCALE=Byelorussian_Belarus;

    test = GETPXLANGUAGE ();
    PUT test;

RUN;
```

This produces the following output:

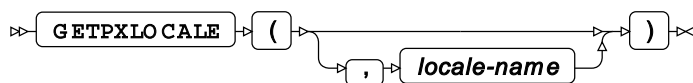
```
be
```

Note:

The `OPTIONS LOCALE` statement changes the **System Options** `LOCALE` setting to Belarus. Unless the system is returned to the original locale setting (in this case, running the DATA step again with the following `change OPTIONS LOCALE=en_GB`, or restarting the server), the Local Server will retain this setting. However, when the WPS Workbench is closed down and restarted, it will revert to the default setting, as specified by the **Startup** and **System Options** `LOCALE` setting.

GETPXLOCALE

Either returns the locale code of the current server, or the locale code of a specified locale.



Return type: Character

locale-name

Optional argument

Type: Character

The name of the locale.

Example – a locale code for language and country

In this example, the locale code for the language and country setting is returned. The result is written to the log.

```
DATA _NULL_;  
  
    test = GETPXLOCALE ();  
    PUT test;  
  
RUN;
```

This produces the following output:

```
en_GB
```

Example – how to find the locale code of a specified locale

In this example, the locale code for Switzerland is returned. Locale values are listed in [this document](#). See *LOCALE Values* [↗](#) (page 37) for more information. There are several language settings for Switzerland (French_Switzerland, German_Switzerland, and Italian_Switzerland). The French speaking language setting is selected, and added to the function before running the DATA step. The result is written to the log.

```
DATA _NULL_;  
  
    test = GETPXLOCALE ('French_Switzerland');  
    PUT test;  
  
RUN;
```

This produces the following output:

```
fr_CH
```

GETPXREGION

Returns a two-character region code for a locale.

➤ GETPXREGION ➤ () ➤

Return type: Character

The locale region code.

Example – a locale region

In this example, the locale region code for the server is returned. The result is written to the log.

```
DATA _NULL_;  
  
    test = GETPXREGION ();  
    PUT test;  
  
RUN;
```

This produces the following output:

```
GB
```

The statements `x = GETOPTION("locale");` and `PUT x;` can be added to the DATA step to provide the locale code for language and country.

Example – how to find a locale region code

In this example, the locale region code for the Costa Rica is returned. For only this example, the locale value for Costa Rica has to be set in the system. Locale values are listed in this document. See [LOCALE Values](#) (page 37) for more information. The setting for Costa Rica is given as `Spanish_CostaRica`. This is then added to the `OPTIONS LOCALE` statement. The result is written to the log.

```
DATA _NULL_;

  OPTIONS LOCALE=Spanish_CostaRica;

  test = GETPXREGION ();
  PUT test;

RUN;
```

This produces the following output:

```
CR
```

Note:

The `OPTIONS LOCALE` statement changes the **System Options** `LOCALE` setting to `Spanish_CostaRica`. Unless the system is returned to the original locale setting (in this case, running the `DATA` step again with the following change `OPTIONS LOCALE=en_GB`, or restarting the server), the Local Server will retain this setting. However, when the WPS Workbench is closed down and restarted, it will revert to the default setting, as specified by the **Startup** and **System Options** `LOCALE` setting.

NLDATE

Returns a date in a locale specific format.

➤ **NLDATE** ➤ (*dt_arg* , *format_string*) ➤

This function has the ability to display the months of the year in another language, provided that the locale setting is first changed to reflect it.

Return type: Character

The requested value.

dt_arg

Type: Numeric

The number of days from the epoch date (January:01:1960:00).

format_string**Type:** Character

The variable.

For example: '%b' for a short three character month name.

The *format_string* is a string, which can contain variables that define date elements. It can consist of a single variable as shown above, or a combination of variables separated by a character or groups of characters. In the following examples, a colon (":") is first used to separate the variables, then in the second example a group of characters are used as separators:

'%B:%W:%Y' for example: November:49:2017

'%#dth of %b' in year %Y' for example: 6th of Dec in year 2017

The complete *format_string* variables for this function consist of the following:

Variable	Description
%#	Remove leading zeros, for example, '%#d' = 5 instead of 05.
%%	Literal '%', for example '20%%' = 20%.
%a	Short weekday. A three character weekday abbreviation.
%A	Long weekday.
%b	Short month. A three character month abbreviation.
%B	Long month.
%C	Long month with blank padding.
%d	Day of month.
%e	Day of month with blank padding.
%F	Long weekday with blank padding.
%j	Julian day. A current day time interval between 1 and 365.25 Earth days in a year, or 1 and 366 Earth days in a Leap Year.
%m	Month number.
%o	Month number with blank padding.
%u	Day number, Monday=1.
%U	Week number, from week 1 of year = first Sunday.
%V	Week number, from week 1 of year = first Monday and Thursday.
%w	Day number, Sunday = 0.
%W	Week number, from week 1 of year = first Monday.
%y	2-digit year.
%Y	4-digit year.

The padding feature takes the largest possible number of spaces, whether in days (that is, 31=2 and Wednesday=9), or months (that is, 12=2, and September=9) and then right adjusts the appropriate output to fit in the equivalent number of spaces.

Example 1

In this example, the `DATE` function is used with the `NLDATE` function to format and write the present month, day, year, and week number. The result is written to the log.

```
DATA _NULL_;  
  
    dt = DATE();  
  
    test = NLDATE(dt, '%B:%d:%Y:%U');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
December:06:2017:49
```

Example 2

In this example, the `DATE` function is used with the `NLDATE` function to format and write the present day, month, and year. The `#` variable is used to remove a leading zero. The result is written to the log.

```
DATA _NULL_;  
  
    dt = DATE();  
  
    test = NLDATE(dt, '%#dth of %b in year %Y');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
6th of Dec in year 2017
```

NLDATM

Returns a datetime in a locale specific format.

⇒ **NLDATM** (*dtm_arg* , *format_string*) ⇐

This function has the ability to display the months of the year in another language, provided that the locale setting is first changed to reflect it.

Return type: Character

The requested value.

dtm_arg

Type: Numeric

The number of milliseconds from the epoch date (January:01:1960:00).

format_string

Type: Character

The variable.

For example: '%b' for a short three character month name.

The *format_string* is a string, which can contain variables that define date elements. It can consist of a single variable as shown above, or a combination of variables separated by a character or groups of characters. In the following examples, a colon (":") is first used to separate the variables, then in the second example a group of characters and colons are used:

'%B:%d:%H:%M:%P' for example: December:06:13:55:pm

'%#dth of %b in year %Y at time %I:%M:%p' for example: 6th of Dec in year 2017 at time 02:05:pm

The complete *format_string* variables for this function consist of the following:

Variable	Description
%#	Remove leading zeros, for example, '%#d' = 6 instead of 06.
%%	Literal '%', for example '20%%' = 20%.
%a	Short weekday. A three character weekday abbreviation.
%A	Long weekday.
%b	Short month. A three character month abbreviation.
%B	Long month.
%C	Long month with blank padding.
%d	Day of month.
%e	Day of month with blank padding.
%F	Long weekday with blank padding.
%j	Julian day. A current day time interval between 1 and 365.25 Earth days in a year, or 1 and 366 Earth days in a Leap Year.
%m	Month number.
%o	Month number with blank padding.
%u	Day number, Monday=1.
%U	Week number, from week 1 of year = first Sunday.
%V	Week number, from week 1 of year = first Monday and Thursday.

Variable	Description
%w	Day number, Sunday = 0.
%W	Week number, from week 1 of year = first Monday.
%y	2-digit year.
%Y	4-digit year.
%H	Hours - 24-hour clock.
%I	Hours - 12-hour clock.
%P or %p	AM/PM.
%S	Seconds.

The padding feature takes the largest possible number of spaces, whether in days (that is, 31=2 and Wednesday=9), or months (that is, 12=2, and September=9) and then right adjusts the appropriate output to fit in the equivalent number of spaces.

Example 1

In this example, the `DATETIME` function is used with the `NLDATM` function, to format and write the present month, day, year, week number, time and whether it is a.m. or p.m. The result is written to the log.

```
DATA _NULL_;  
  
    dt = DATETIME();  
  
    test = NLDATM(dt, '%B:%d:%Y:%U:%I:%M:%S:%P');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
December:06:2017:46:01:55:26:pm
```

Example 2

In this example, the `DATETIME` function is used with the `NLDATM` function, to format and write the present day, month, year, time and whether it is a.m. or p.m. The `#` variable is used to remove a leading zero. The result is written to the log.

```
DATA _NULL_;  
  
    dt = DATETIME();  
  
    test = NLDATM(dt, '%#dth of %b in year %Y at time %I:%M:%p');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
6th of Dec in year 2017 at time 02:25:pm
```

NLTIME

Returns a time or datetime in a locale specific format for hour, minute, second, and AM/PM.

➤ **NLTIME** ➤ (*datetime_arg* , *format_string*) ➤

Return type: Character

The requested time.

datetime_arg

Type: Numeric

The number of milliseconds from midnight (00:00:00).

format_string

Type: Character

The variable.

For example: '%I' for a current 12-hour clock value.

The *format_string* is a string, which can contain variables that define time elements. It can consist of a single variable as shown above, or a combination of variables separated by a character or groups of characters. In the following examples, a colon (":") is first used to separate the variables, then in the second example a group of characters and colons are used:

'%H:%M:%S:%P' for example: 12:04:10:pm

'Hours: %H Minutes: % Minutes: % Seconds: %S (AM/PM) %P' for example: Hours 15 Minutes: 27 Seconds: 52 (AM/PM) pm

The complete *format_string* variables for this function consist of the following:

Variable	Description
%#	Remove leading zeros, for example, '%#I' = 9 instead of 09.
%%	Literal '%', for example '20%%' = 20%.
%H	Hours - 24-hour clock.
%I	Hours - 12-hour clock.
%M	Minutes.
%P or %p	AM/PM.
%S	Seconds.

Example 1

In this example, the `TIME` function is used with the `NLTIME` function to format and write the time at which the `time` function was invoked. Colons are used as separators. The result is written to the log.

```
DATA _NULL_;  
  
    dt = TIME();  
  
    test = NLTIME(dt, '%I:%M:%S:%p');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
12:40:28:pm
```

Example 2

In this example, the `DATETIME` function is used with the `NLTIME` function to format and write the time at which the `datetime` function was invoked. Characters are used as separators. The result is written to the log.

```
DATA _NULL_;  
  
    dt = DATETIME();  
  
    test = NLTIME(dt, 'Hours: %H Minutes: %M Seconds: %S (AM/PM) %P');  
  
    PUT test;  
  
RUN;
```

This produces the following output:

```
Hours: 15 Minutes: 34 Seconds: 12 (AM/PM) pm
```

Regular expression functions and `CALL` routines

Find and manipulate strings using regular expressions.

You can find and replace characters and strings using the flexibility of pattern-matching provided by regular expressions. The regular expression functionality provided in these functions and call routines is based on the Perl regular expression syntax.

In many of the functions and `CALL` routines the regular expression can be specified in two ways:

- Explicitly within the function or `CALL`; for example, `prxmatch('/(Limited) | (LTD) | (Ltd)+/', lr)`

- As a compiled regular expression selected by an identifier. Regular expressions can be compiled using the `PRXPARSE` function, which generates an identifier that can be used in other functions and `CALL` routines. For example,

```
rxid=prxparse('(/[Oo]range)|([Aa]pple)|([Bb]anana)|([Gg]rapefruit)/');
```

would compile the regular expression and return an identifier to `rxid`. The identifier for that regular expression could then be used in another function or `CALL` routine. For example:

```
rxm = PRXMATCH(rxid,'Banana, Ghana, 3.6, Grapefruit');
```

If the regular expressions are explicitly specified, the source code is easier to read as you can see the structure of regular expressions wherever they occur. However, if regular expressions are specified in this way they will be processed and compiled every time they are encountered. If a regular expression is commonly used, or used to manipulate multiple observations, explicit specification might lead to unacceptable processing costs as it is compiled many time.

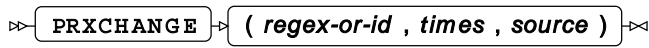
If you use `PRXPARSE` to compile a regular expression, other functions and `CALL` routines can refer to that compiled expression through an identifier. If that regular expression is required, again reference can be made to the identifier, saving time required for compilation.

<code>PRXCHANGE</code> ↗	1915
Returns the string that results from finding and replacing substrings in a specified source string.	
<code>PRXMATCH</code> ↗	1916
Returns the position of the first character that matches a pattern defined using a regular expression.	
<code>PRXPAREN</code> ↗	1917
Returns which substring in a specified list occurs in a specified string.	
<code>PRXPARSE</code> ↗	1919
Compiles a regular expression for use with other functions or <code>CALL</code> routines that require regular expressions.	
<code>PRXPOSN</code> ↗	1920
Return the contents of a Perl-style capture buffer.	
<code>CALL PRXCHANGE</code> ↗	1922
Returns the string that results from finding and replacing substrings in a specified source string, using a regular expression.	
<code>CALL PRXDEBUG</code> ↗	1924
Switch on debugging for regular expressions.	
<code>CALL PRXFREE</code> ↗	1926
Free a regular expression identifier.	
<code>CALL PRXNEXT</code> ↗	1927
Returns the start and end position of the next substring in a specified source string that matches a specified regular expression.	
<code>CALL PRXPOSN</code> ↗	1929
Returns the position and length of a Perl-style capture buffer.	

CALL PRXSUBSTR ↗	1931
Returns the position of the first occurrence of a substring in a source string, using a regular expression to specify the string to be found.	

PRXCHANGE

Returns the string that results from finding and replacing substrings in a specified source string.



Substrings are found and replaced using a regular expression. For example, if your data contains details of companies, the 'Limited' in their name might be represented in various ways, such as 'Limited', 'LTD', or 'Ltd.', that you might prefer to represent simply as 'Ltd'.

Return type: Character

regex-or-id

Type: Character or numeric value

A regular expression, or an identifier generated by the [PRXPARSE](#) [↗](#) (page 1919) function (which generates an identifier for a compiled regular expression).

times

Type: Numeric

The number of matching patterns in the string to be replaced. Use -1 to replace all matching patterns. If you specify 0, no matching patterns are replaced.

source

Type: Character

The string to be modified.

Example

In this example, observations are searched for the word `Bike`. The result is written to the log.

```
DATA _NULL_;
  INFILE DATALINES END=LASTOBS;

  INPUT a $50.;
  NAME = right(a);

  IF _N_ = 1 THEN DO;
    rxid = PRXPARSE('s/Bike/Bicycle/');
    RETAIN rxid;
  END;

  xc = PRXCHANGE(rxid,-1, name);
  result = LEFT(xc);
  PUT result;

  IF LASTOBS THEN CALL PRXFREE(id);

DATALINES;
  Magnificent Bike Company Limited
  London Bike LTD.
  Old Penny Farthing Ltd
  Racing Bikes of Reading and London Lmted.
;
```

This produces the following output:

```
Magnificent Bicycle Company (Bicycles) Limited
Magnificent Bicycle Company (Misc.) Limited
London Bicycle Limited.
Old Penny Farthing Ltd
Racing Bicycles of Reading and London Lmted.
```

PRXMATCH

Returns the position of the first character that matches a pattern defined using a regular expression.

➤ **PRXMATCH** ➤ (*regex-or-id* , *source*) ➤

Return type: Numeric

If no matching pattern is found, 0 (zero) is returned.

regex-or-id

Type: Character or numeric value

A regular expression, or an identifier generated by the `PRXPARSE` [\(page 1919\)](#) function (which generates an identifier for a compiled regular expression).

source**Type:** Character

The source string.

Example

In this example, observations are searched for strings similar to `Limited`, such as `LTD` and `Ltd`, and returns the first position in the observation at which it was found. The result is written to the log.

```
DATA _NULL_;
  INFILE DATALINES;
  INPUT a $45.;
  IF _N_ = 1 THEN DO;
    PUT "The positions of the matching strings are:";
  END;
  lr = TRIM(a);
  xm = PRXMATCH('/(Limited)|(LTD)|(Ltd)+/', lr);
  PUT xm;
DATALINES;
Magnificent Bikes Company
Magnificent Bikes (Subs) Limited
London Bike LTD.
Old Penny Farthing Inc.
Racing Bikes of Reading and London Limited.
;
```

This produces the following output:

```
The positions of the matching strings are:
0
26
13
0
36
```

PRXPAREN

Returns which substring in a specified list occurs in a specified string.

➤ **PRXPAREN** ➤ (*regex-id*) ➤

Searches the source string supplied to the last `PRXMATCH` for one of a list of parenthesised search strings in a regular expression, and returns the position of the first matching parenthesised element. For example, you might want to find which of the strings `Sugar`, `Banana`, `Ghana`, and `India` occurs in a source string; this function enables you to do that. It returns the ordinal position of the first string in the list that matches text in the source. In this example, if the regular expression listed the search patterns in the order shown above, and the string to be searched was `Tea, Ghana, 3.6`, the function would return 2, as the first string to be found is `Ghana` which is at position two in the list.

Return type: Numeric

regex-id

Type: Numeric

A regular expression identifier generated by the `PRXPARSE` [↗](#) (page 1919) function (which generates an identifier for a compiled regular expression).

This function must be used with the `PRXPARSE` function, which generates the identifier for a compiled regular expression, and with the `PRXMATCH` function.

Note:

The regular expression search for parenthesised strings must be separated by the OR (|) symbol for the function to return the correct result; otherwise 0 is returned. Using the example above, the regular expression would need to be specified as `/(Sugar) | (Banana) | (Ghana) | (India)/`.

Basic example

In this example, `PRXPARSE` is used to generate an identifier for a parsed regular expression, and this is passed to the `PRXPAREN` function. The result is written to the log.

```
DATA _NULL_;
  rxid = PRXPARSE('(/[Oo]range) | ([Aa]pple) | ([Bb]anana) | ([Gg]rapefruit)/');
  result = PRXPAREN(rxid);
  PUT "The result is: " result;
RUN;
```

This produces the following output:

```
The result is: .
```

The output contains the missing value (.) as `PRXPAREN` had no result from a previous `PRXMATCH`.

Example – using `PRXMATCH`

In this example, `PRXPARSE` is used to generate an identifier for a parsed regular expression, and this is passed to both the `PRXMATCH` and `PRXPAREN` functions. The result is written to the log.

```
DATA _NULL_;
  rxid = PRXPARSE('(/[Oo]range) | ([Aa]pple) | ([Bb]anana) | ([Gg]rapefruit)/');
  rxm = PRXMATCH(rxid, 'Banana, Ghana, 3.6, Grapefruit');
  result = PRXPAREN(rxid);
  PUT "The result of the PRXMATCH is: " rxm;
  PUT "The result of the PRXPAREN is: " result;
RUN;
```

This produces the following output:

```
The result of the PRXMATCH is: 1
The result of the PRXPAREN is: 3
```

In the `PRXMATCH` function, the string `Banana` matches `Banana` in the source at the first position, so `rxm` is set to 1. The function has also read the source string into memory, enabling `PRXPAREN` to operate on it using the regular expression. `PRXPAREN` returns 3, as `Banana` in the source string matches the third search string (counting from the left) in the regular expression. Only the first occurrence of a search string is found, so the match with `Grapefruit` in the source string is ignored.

PRXPARSE

Compiles a regular expression for use with other functions or `CALL` routines that require regular expressions.

➤ `PRXPARSE` ➤ `(regex)` ➤

Returns an identifier that provides the location of the compiled expression that can then be supplied in place of a regular expression. The returned identifier can be used as many times as required until cleared using `CALL PRXFREE`.

Each invocation of this function returns a different identifier that is unique to the regular expression compiled by the function; typically, this identifier is a number that increases by one for each invocation.

For some of the regular-expression functions, such as `PRXPAREN`, an identifier *must* be supplied, rather than a regular expression.

Return type: Numeric

regex

Type: Character

The regular expression to be interpreted.

Basic example

In this example, the regular expressions are interpreted and identifiers created for them, which can be used in other functions. The result is written to the log.

```
DATA _NULL_;
  rxd1 = PRXPARSE('/(Banana)|(Orange)|(Apple)|(Grapefruit)');
  rxd2 = PRXPARSE('/(Wheat)|(Barley)|(Maize)|(Sorghum)');
  PUT "The regular expression identifier is: " rxd1;
  PUT "The regular expression identifier is: " rxd2;
RUN;
```

This produces the following output:

```
The regular expression identifier is: 1
The regular expression identifier is: 2
```

Example – creating identifier for use by another function

In this example, an identifier is created for a regular expression, which is then passed to the `PRXCHANGE` function. The result is written to the log.

```
DATA _NULL_
  INFILE DATALINES END=LASTOBS;
  INPUT a $50.;
  name = RIGHT(a);
  IF _N_ = 1 THEN DO;
    rxid = PRXPARSE('s/Bike/Bicycle/');
    RETAIN rxid;
  END;
  xc = PRXCHANGE(rxid,-1, name);
  result = LEFT(xc);
  PUT result;
  IF LASTOBS THEN CALL PRXFREE(id);
DATALINES;
Magnificent Bike Company Limited
London Bike LTD.
Old Penny Farthing Ltd
Racing Bikes of Reading and London Lmtl.
;
```

The regular expression searches a source string for the string `Bike` and then exchanges them for the text `Bicycle`. The expression is interpreted by `PRXPARSE`, and an identifier for it is created. The identifier is then used in the `PRXCHANGE` function; the regular expression is evaluated as if it had been entered into the function.

This produces the following output:

```
Magnificent Bicycle Company Limited
London Bicycle LTD.
Old Penny Farthing Ltd
Racing Bicycles of Reading and London Lmtl.
```

PRXPOSN

Return the contents of a Perl-style capture buffer.

```
PRXPOSN ( regex-id , capture-buffer , source )
```

Returns the contents of a specified Perl-style capture buffer; the substrings to be found and the capture buffers are defined using a regular expression. Before you call `PRXPOSN` to obtain the text contents of the capture buffers:

- Compile the regular expression using the `PRXPARSE` [\(page 1919\)](#) function, which returns an identifier that can be used by this function.
- Call `PRXMATCH`, using the regular expression identifier supplied by `PRXPARSE`, to obtain the text contents of the capture buffers, and establish their locations and lengths.

The string *source* does not have to be the same as the string previously provided to `PRXMATCH`. The string in `PRXMATCH` could be used to find the locations of keywords in a sample string, and `PRXPOSN` could then be used to extract text from the same position in different strings.

Return type: Character

regex-id

Type: Numeric

A regular expression identifier.

capture-buffer

Type: Numeric

The index number of a capture-buffer. Capture buffers are numbered from 1 to however many have been defined in the regular expression. Capture buffer 0 (zero) contains all strings in all capture buffers.

source

Type: Character

The source string to be examined by the regular expression.

Basic example

In this example, the regular expression is interpreted and an identifier created for it using `PRXPARSE`. The identifier is used in the `PRXPOSN` function. The result is written to the log.

```
DATA _NULL_;
  a = "Magnificent Bike Company (Misc) Limited";
  id = PRXPARSE('/((Bike) (Company)) (\(Misc\))/');
  match = PRXMATCH(id,a);
  result0 = PRXPOSN(id,0,a);
  result1 = PRXPOSN(id,1,a);
  result2 = PRXPOSN(id,2,a);
  result3 = PRXPOSN(id,3,a);
  result4 = PRXPOSN(id,4,a);
  PUT result0;
  PUT result1;
  PUT result2;
  PUT result3;
  PUT result4;
RUB;
```

This produces the following output:

```
Bike Company (Misc)
Bike Company
Bike
Company
(Misc)
```

The `PRXPOSN` function uses the positions of the substrings generated by `PRXMATCH` to identify the substrings. In the example, `PUT` is used to write the strings found in each capture buffer, including capture buffer 0, which contains all the strings. The order in which the strings are assigned to capture buffers depends on the bracketing.

Example – different strings in `PRXMATCH` and `PRXPOSN`

In this example, the regular expression is interpreted and an identifier created for it using `PRXPARSE`. The identifier is used in the `PRXMATCH` function match identify the position and length of the strings in the capture buffers. The positions are then used in `PRXPOSN` to return strings at those positions with those lengths. The result is written to the log.

```
DATA _NULL_;
  a = "Magnificent Bike Company (Misc) Limited";
  id = PRXPARSE('/((Bike) (Company)) (\(Misc\))/');
  match = PRXMATCH(id,a);
  result0 = PRXPOSN(id,0,a);
  result1 = PRXPOSN(id,1,a);
  result2 = PRXPOSN(id,0,'Magnificent Cars Company (Misc) Limited');
  result3 = PRXPOSN(id,0,'Magnificent Tiger Company (Misc) Limited');
  result4 = PRXPOSN(id,2,'Magnificent Tiger Company (Misc) Limited');
  PUT result0;
  PUT result1;
  PUT result2;
  PUT result3;
  PUT result4;
RUN;
```

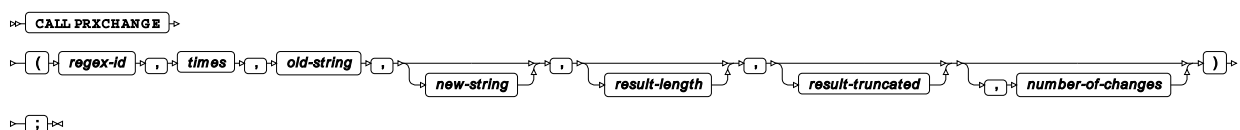
This produces the following output:

```
Bike Company (Misc)
Bike Company
Cars Company (Misc)
Tiger Company (Misc
Tige
```

The `PRXPOSN` function uses the positions of the substrings generated by `PRXMATCH` to identify the substrings. In the example, `PUT` is used to write the strings found in capture buffers 0 and 1 for the string supplied to `PRXMATCH`. The locations found by the regular expression in the `PRXMATCH` are then applied to different strings in subsequent lines. Because the word `Tiger` is longer than `Bike`, the information returned is truncated to match the length of the capture buffer.

CALL PRXCHANGE

Returns the string that results from finding and replacing substrings in a specified source string, using a regular expression.



The modified string is returned to a variable specified as an argument to the routine. For example, if your data contains details of companies, the 'Limited' in their name might be represented in various ways, such as 'Limited', 'LTD', or 'Ltd.', that you might prefer to represent simply as 'Ltd'.

This routine also optionally returns the length of the modified string, the number of changes made, and indicates whether the modified string has been truncated as a result of the modifications.

This call routine is similar to the `PRXCHANGE` [↗](#) (page 1915) function.

regex-id

Type: Numeric

A regular expression identifier generated by the `PRXPARSE` [↗](#) (page 1919) function (which generates an identifier for a compiled regular expression).

times

Type: Numeric

The number of matching patterns in the string to be replaced. Use `-1` to replace all matching patterns. If you specify `0`, no matching patterns are replaced.

old-string

Type: Character

The source string to be modified.

new-string

Optional argument

Type: Character

The variable into which the resulting string is returned.

result-length

Optional argument

Type: Numeric

A value returned by this routine. The length of the modified string.

result-truncated

Optional argument

Type: Numeric

A value returned by this routine, indicating whether or not *new-string* was truncated to fit the length specified for it. This value is `1` if *new-string* has been truncated, `0` otherwise.

number-of-changes

Optional argument

Type: Numeric

A value returned by this routine. The number of changes made to *old-string* to create *new-string*.

Example

In this example, observations are searched for the word `Bike`, which are replaced with `Bicycle` in all cases. The result is written to the log.

```
DATA _NULL_;
  INFILE DATALINES END=LASTOBS;
  INPUT a $50.;
  LENGTH ns $40;
  IF _N_ = 1 THEN DO;
    rxid = PRXPARSE('s/Bike/Bicycle/');
    RETAIN rxid;
  END;
  CALL PRXCHANGE(1, -1, a, ns, rl, rt, noc);
  PUT ns " " rl " " rt " " noc;
  IF LASTOBS THEN CALL PRXFREE(id);
DATALINES;
Magnificent Bike Company (Bikes) Limited
Magnificent Bike Company (Misc.) Limited
London Bike Limited.
Old Penny Farthing Ltd
Racing Bikes of Reading and London Lmted.
;
```

This produces the following output:

```
Magnificent Bicycle Company (Bicycles) L 40 1 2
Magnificent Bicycle Company (Misc.) Limi 40 1 1
London Bicycle Limited. 23 0 1
Old Penny Farthing Ltd 22 0 0
Racing Bicycles of Reading and London Lm 40 1 1
```

In this example, `-1` has been specified for the *times* argument, which means all matching patterns will be replaced. In the first result, therefore, `Bike` has been replaced with `Bicycle` twice (as shown by the `2` returned in *nt*). In the first two results and the last result, replacing `Bike` with `Bicycle` has resulted in a string longer than the length specification for that string. The value `1` has therefore been returned in *rt*. In the third and fourth results, the string after substitution is shorter than the length specification; `0` is therefore returned in *rt*.

CALL PRXDEBUG

Switch on debugging for regular expressions.

```
➤ CALL PRXDEBUG ➤ ( on-off ) ➤ ; ➤
```

Switches on and off debugging for regular expressions. If switched on, messages are written to the log about the operation of regular expression functions.

on-off**Type:** Numeric

An integer that switches on or off regular expression debugging. 0 switches debugging off; any other value switches it on.

Example

In this example, the regular expressions are compiled and identifiers created for each of them, which can then be used in other functions. The result is written to the log.

```
DATA _NULL_;
  INFILE DATALINES END=LASTOBS;
  INPUT a $50.;
  LENGTH ns $50;
  CALL PRXDEBUG(1);
  IF _N_ = 1 THEN DO;
    rxid = PRXPARSE('s/Bike/Bicycle/');
    RETAIN rxid;
  END;
  rxid = PRXPARSE('s/Bike/Car/');
  CALL PRXCHANGE(1, -1, a, ns, rl, rt, noc);
  PUT ns " " rl " " rt " " noc;
  IF LASTOBS THEN CALL PRXFREE(id);
DATALINES;
Magnificent Bike Company (Bikes) Limited
Magnificent Bike Company (Misc.) Limited
London Bike LTD.
Old Penny Farthing Ltd
Racing Bikes of Reading and London Lmtd.
;
```

CALL PRXDEBUG is set to 1, so debugging is switched on. Debugging messages are returned to the log for each record; in this example, each series of messages is followed by the results written to the log by the PUT statement.

This produces the following messages and output for the first input record for this example:

```
PRXDEBUG: CALL PRXDEBUG:
PRXDEBUG: Regular expression debug setting set to ON

PRXDEBUG: PRXPARSE:
PRXDEBUG: Parsing Regular expression s/Bike/Car/
PRXDEBUG: Compiling regular expression pattern Bike
PRXDEBUG: Replacement text is: Car
PRXDEBUG: Delimiter is: /
PRXDEBUG: Compilation successful
PRXDEBUG: Regular expression successfully parsed

PRXDEBUG: CALL PRXCHANGE:
PRXDEBUG: Retrieving Parsed Regular Expression from ID: 1
PRXDEBUG: Retrieved Parsed Regular Expression
PRXDEBUG: Replacing in 'Magnificent Bike Company (Bikes) Limited' every time it
occurs
PRXDEBUG: Result is 'Magnificent Car Company (Cars) Limited'
Magnificent Car Company (Cars) Limited 38 0 2
```

CALL PRXFREE

Free a regular expression identifier.

➤ `CALL PRXFREE` ➤ `(regex-id)` ➤ `;` ➤

Frees a specified regular expression identifier of its current regular expression. This effectively clears the regular expression from memory; any further call to that identifier by a regular expression function or call routine will result in an error until the identifier is reassigned by a later invocation of `PRXPARSE`.

regex-id

Type: Numeric

A regular expression identifier.

Basic example

In this example, the regular expression is compiled and an identifier created using `PRXPARSE`. The identifier is used in the `CALL PRXCHANGE` function. The `CALL PRXFREE` option is then used to free the regular expression identifier.

```
DATA _NULL_;
  INFILE DATALINES END=LASTOBS;
  INPUT a $50.;
  LENGTH ns $50;
  IF _N_ = 1 THEN DO;
    RETAIN ID;
    id = PRXPARSE('s/Bike/Car/');
  END;
  CALL PRXCHANGE(id, -1, a, ns, rl, rt, noc);
  PUT ns " " rl " " rt " " noc;
  IF LASTOBS THEN CALL PRXFREE(id);
DATALINES;
Magnificent Bike Company (Bikes) Limited
Magnificent Bike Company (Misc.) Limited
London Bike LTD.
Old Penny Farthing Ltd
Racing Bikes of Reading and London Lmted.
;
```

Example – error returned because identifier prematurely freed

In this example, the regular expression is compiled and an identifier created when the first observation in the dataset is read. The `CALL PRXFREE` option is then used to free the regular expression identifier before the `CALL PRXCHANGE` function is called for the next observation; the `CALL PRXCHANGE` does not, therefore, have a valid value for its identifier.

```
DATA _NULL_;
  INFILE DATALINES END=LASTOBS;
  INPUT a $50.;
  LENGTH ns $50;
  IF _N_ = 1 THEN DO;
    RETAIN ID;
    id = PRXPARSE('s/Bike/Car/');
  END;
  CALL PRXCHANGE(id, -1, a, ns, rl, rt, noc);
  PUT ns " " rl " " rt " " noc;
  CALL PRXFREE(id);
DATALINES;
Magnificent Bike Company (Bikes) Limited
Magnificent Bike Company (Misc.) Limited
London Bike LTD.
Old Penny Farthing Ltd
Racing Bikes of Reading and London Lmtd.
;
```

Because `CALL PRXCHANGE` has no valid identifier, an error message similar to the following is generated for each line of data subsequent to the first.

```
ERROR: Argument 1 to the function PRXCHANGE must be a value returned by PRXPARSE for
a valid
      pattern
NOTE: Argument 1 to function PRXCHANGE at line 173 column 7 is invalid
```

CALL PRXNEXT

Returns the start and end position of the next substring in a specified source string that matches a specified regular expression.

```
⇒ CALL PRXNEXT ⇒ ( regex-id , start-position , stop-position , source , position , length ) ⇒ ; ⇒
```

You must define the position in the source string at which to start and stop searching. The regular expression must first be compiled using the `PRXPARSE` function, which returns an identifier that can be used in this function.

The start and end position are returned to variables you specify as arguments to this routine.

regex-id

Type: Numeric

A regular expression identifier.

start-position

Type: Numeric

The position in the *source* string at which to start searching for the substring.

stop-position

Type: Numeric

The position in the *source* string at which to stop searching for the substring.

source

Type: Character

The string to be examined for a pattern.

position

Type: Numeric

The position in *source* at which the substring is found.

length

Type: Numeric

The length of the substring found.

Basic example

In this example, the regular expression is compiled and an identifier created for it using `PRXPARSE`. The identifier is used in the `CALL PRXNEXT` function. The result is written to the log.

```
DATA _NULL_;  
  id = PRXPARSE('/Bike/');  
  a = 'Magnificent Bike Company (Bikes) Limited';  
  CALL PRXNEXT(id, 1, length(a), a, pos, len);  
  PUT "String found at: " pos;  
  PUT "String length is: " len;  
RUN;
```

This produces the following output:

```
String found at: 13  
String length is: 4
```

The function starts searching the source string at the first character position, and stops at last character. These results are the position and length of the first occurrence of the substring `Bike`.

Example – specifying start and end point

In this example, the regular expression is compiled and an identifier for it created using `PRXPARSE`. The identifier is used in the `CALL PRXCHANGE` function. The result is written to the log.

```
DATA _NULL_;
  id = PRXPARSE('/Bike/');
  a = 'Magnificent Bike Company (Bikes) Limited';
  CALL PRXNEXT(id, 20, 35, a, rl, rt);
  PUT "String found at: " rl;
  PUT "String found at: " rt;
RUN;
```

This produces the following output:

```
String found at: 27
String length is: 4
```

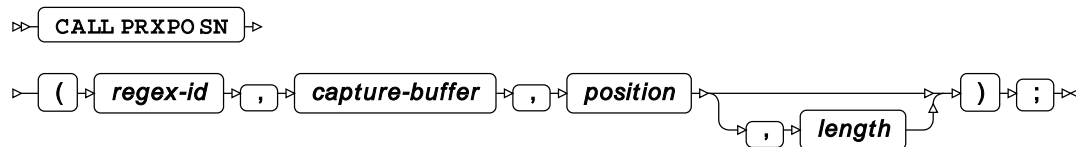
The function starts searching the source string at character position 20, and stops at position 35. The next occurrence of the substring `Bike` is now found at position 27.

Note:

If the end position for the search also terminates the string to be found, the string will not be found. In the example above, if the search had started at character 20 and ended at character 29, the range would contain only the string `pany (Bik`, which contains no match for `Bike`.

CALL PRXPOSN

Returns the position and length of a Perl-style capture buffer.



This call routine is similar to the `PRXPOSN` [\(page 1920\)](#) function, except this routine returns the position and length of the capture buffers, rather than the contents of those buffers.

The substrings to be found and the capture buffers are defined using a regular expression. Before you use `CALL PRXPOSN` to obtain the text contents of the capture buffers:

- The regular expression must first be compiled using the `PRXPARSE` [\(page 1919\)](#) function, which returns an identifier that can be used by this function.
- Call `PRXMATCH`, using the regular expression identifier supplied by `PRXPARSE`, to obtain the text contents of the capture buffers, and establish their locations and lengths.

regex-id

Type: Numeric

A regular expression identifier.

capture-buffer

Type: Numeric

The index number of a capture-buffer. Capture buffers are numbered from 1 to n , where n is the number of buffers defined in the regular expression. Capture buffer 0 (zero) contains all strings in all capture buffers (that is, the entire search string).

position

Type: Numeric

A value returned by this routine. The position at which the string in *capture-buffer* was found in the source.

length

Optional argument

Type: Numeric

A value returned by this routine. The length of the string in *capture-buffer*.

Example

In this example, the regular expression is compiled and an identifier created for it using `PRXPARSE`. The identifier is used in the `CALL PRXPOSN` function. The result is written to the log.

```
DATA _NULL_;
  id = PRXPARSE('/(Bike) (Company) (Misc)/');
  a = "Magnificent Bike Company (Misc) Limited";
  match = PRXMATCH(id,a);
  put "PRXMATCH matches first at: " match;
  put "-----";
  CALL PRXPOSN (id,0, start, length);
  put "The strings captured in all buffers start at: " start;
  put "The strings captured in all buffers are: " length "characters long";
  put "-----";
  CALL PRXPOSN (id, 1, start, length);
  put "String in capture buffer 1 found at : " start;
  put "Length of string in capture buffer 1 : " length;
  put "-----";
  CALL PRXPOSN (id, 2, start, length);
  put "String in capture buffer 2 found at : " start;
  put "Length of string in capture buffer 2 : " length;
  put "-----";
  CALL PRXPOSN (id, 3, start, length);
  put "String in capture buffer 3 found at : " start;
  put "Length of string in capture buffer 3 : " length;
RUN;
```


This produces the following output:

```
PRXMATCH matches first at: 13
-----
The strings captured in all buffers start at: 13
The strings captured in all buffers are: 19 characters long
-----
String in capture buffer 1 found at   : 13
Length of string in capture buffer 1 : 4
-----
String in capture buffer 2 found at   : 18
Length of string in capture buffer 2 : 7
-----
String in capture buffer 3 found at   : 26
Length of string in capture buffer 3 : 6
```

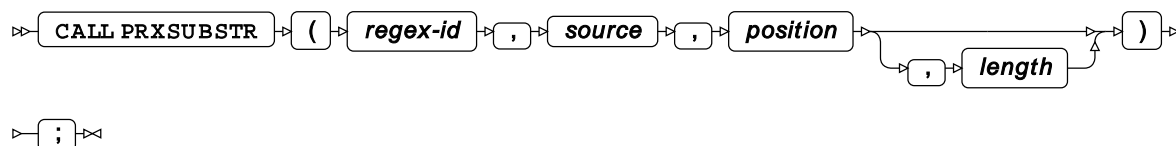
The PRXMATCH function has been used to ensure the capture buffer location and extents have been identified for the regular expression identifier.

The capture buffer 0 contains the strings starting at position 13 (the same as found by PRXMATCH, and in the first capture buffer in CALL PRXPOSN). The string length returned for capture buffer 0 is nineteen characters, as it is the length of all strings captured to buffers 1 to 3 (the entire search pattern); in this case `Bike Company (Misc)`.

The lengths and positions of each string found in each capture buffer are then written to the log.

CALL PRXSUBSTR

Returns the position of the first occurrence of a substring in a source string, using a regular expression to specify the string to be found.



The regular expression must first be interpreted using the PRXPARSE function, which returns an identifier that can be used in this function.

regex-id

Type: Numeric

A regular expression identifier.

source

Type: Character

The string to be examined for a pattern.

position**Type:** NumericThe position in *source* at which the substring is found.***length***

Optional argument

Type: Numeric

The length of the substring found.

Example

In this example, the regular expression is interpreted and an identifier created for it using `PRXPARSE`. The identifier is used in the `CALL PRXSUBSTR` function. The result is written to the log.

```
DATA _NULL_;
  a = "Magnificent Bike Company (Bikes) Limited";
  id = PRXPARSE('/Bike/');
  CALL PRXSUBSTR (id,a, pos, len);
  PUT "String found at : " pos;
  PUT "Length of string: " len;
RUN;
```

This produces the following output:

```
String found at : 13
Length of string: 4
```

Sequence manipulation functions

Operate on sequences (lists) of items in variables.

CHOOSEC ↗	1933
Returns the string found at a specified position in a list of comma-separated strings.	
CHOOSEN ↗	1934
Returns the number found at a specified location in a list of numbers.	
COALESCEC ↗	1935
Returns the first string that contains characters other than all spaces or null.	
COALESCE ↗	1936
Returns the first non-missing value in a list of numeric values.	
WHICHC ↗	1937
Returns the position of the first string in a list of strings that matches a specified string.	
WHICHN ↗	1937
Returns the position of a specified argument in a list of numeric or missing values.	

CALL SORTC ↗	1938
Sorts a list of strings.	
CALL SORTN ↗	1940
Returns the values in a list of numeric variables sorted in ascending order.	

CHOOSEC

Returns the string found at a specified position in a list of comma-separated strings.



Return type: Character

value

Type: Numeric

The ordinal position of the string in the list of strings. This should be an integer. For example, if there are five strings, the leftmost is number 1, and the rightmost number 5; to select the third string in the list, specify 3 for this argument. If the number is negative, counting progresses from right to left.

item

Type: Character

A string, or a variable containing a string.

A missing value is returned if *value* is greater than the number of items in the list.

Example – returning string found at specified position

In this example, the function is used to find the second string in the list of strings provided to the function. The result is written to the log.

```

DATA _NULL_;
  string1="Company";
  result = CHOOSEC(2,"Magnificent","Bike", string1,"London");
  PUT "The string found is: " result;
RUN;
  
```

This produces the following output:

```

The string found is: Bike
  
```

Example – returning string contained in variable

In this example, the function is used to find the third string in the list of strings provided to the function. In this example, the third string is contained in a variable. The result is written to the log.

```
DATA _NULL_;  
  string1="Company";  
  result = CHOOSESEC(3,"Magnificent","Bike", string1,"London");  
  PUT "The string found is: " result;  
RUN;
```

This produces the following output:

```
The string found is: Company
```

The function returns the third string in the list, which in this example is `Company`, the string contained in the variable `string1`.

CHOOSESEC

Returns the number found at a specified location in a list of numbers.



Return type: Numeric

value

Type: Numeric

The ordinal position of the number in the list of numbers. This should be an integer. For example, if there are five numbers, the leftmost is number 1, and the rightmost is number 5; to select the third number in the list, specify 3 for this argument. If the number is negative, counting progresses from right to left.

item

Type: Numeric

A number.

A missing value is returned if *value* is greater than the number of items in the list.

Example – returning number found at specified position

In this example, the function is used to find the second number in the list of numbers provided to the function. The result is written to the log.

```
DATA _NULL_;  
  var1=10;  
  result = CHOSEN(2, 1, 100, var1, 7, 12, 9);  
  PUT "The number found is: " result;  
RUN;
```

This produces the following output:

```
The number found is: 100
```

Example – returning number contained in variable

In this example, the function is used to find the fourth number in the list of numbers provided to the function. The result is written to the log.

```
DATA _NULL_;  
  var1=10;  
  result = CHOSEN(4, 1, 100, var1, 7, 12, 9);  
  PUT "The number found is: " result;  
RUN;
```

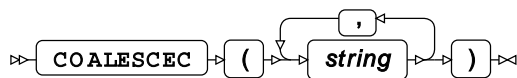
This produces the following output:

```
The number found is: 100
```

The function has returned the number at the fourth position in the list, which is contained in the variable `var1`.

COALESCEC

Returns the first string that contains characters other than all spaces or null.



Return type: Character

string

Type: Character

A string to be examined.

Example

In this example, the function searches the strings provided to it for the first that does not contain only spaces or is null. The result is written to the log.

```
DATA _NULL_;  
  empty="  ";  
  result = COALESCEC(' ', ' ', empty, ' World Programming');  
  PUT "The resulting string is: " result;  
RUN;
```

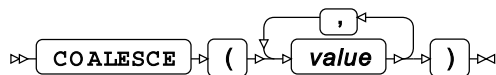
This produces the following output:

```
The resulting string is: World Programming
```

The first string provided to the function is null, the second string is all spaces, and the variable `empty` also contains all spaces; the function therefore returns `World Programming`.

COALESCE

Returns the first non-missing value in a list of numeric values.



Return type: Numeric

value

Type: Numeric

The value to be evaluated.

If all specified values are missing values, a missing value is returned.

Example

In this example, the first non-missing value is returned. The result is written to the log.

```
DATA _NULL_;  
  a = COALESCE("four", ., "4", 1, -2, 3, ., "");  
  PUT a=;  
RUN;
```

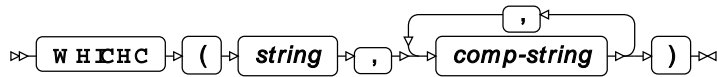
This produces the following output:

```
a=4
```

The argument value `"4"` has been converted into a number, but the argument value `"four"` is considered missing.

WHICHC

Returns the position of the first string in a list of strings that matches a specified string.



Identifies the first string in a list of strings that matches the string you specify, and returns as a number the ordinal position of the matched string in the list of strings. The strings for which to search are specified as one or more comma-separated strings. For example, if the list of strings has four items, and the third string matches, then 3 is returned. If no matching string is found, 0 is returned.

Return type: Numeric

string

Type: Character

The string you want to find in a list of other strings.

comp-string

Type: Character

A string to be matched with *string*.

Example

In this example, the function is used to find the string `Bike` in a list of strings. The result is written to the log.

```
DATA _NULL_;  
  result1 = WHICHC('Bike','Bice', 'Boke', 'Bike');  
  PUT "The string is number " result1 "in the list";  
RUN;
```

This produces the following output:

```
The string is number 3 in the list
```

WHICHN

Returns the position of a specified argument in a list of numeric or missing values.



Searches for the value of *argument* in the list of values that follow it, including missing values. If a value equal to *argument* is encountered, its index in the value list is returned, otherwise 0 is returned.

Return type: Numeric

argument

Type: Numeric

The value to look for in the list that follows.

If the argument contains a missing value, a missing value is returned.

value

Type: Numeric

The value to be evaluated.

Note:

Here positions of all values are counted, including missing values.

Examples

In these examples, the position of a specified argument is returned. The results are written to the log.

```
DATA _NULL_;  
  n1 = WHICHN(4.1, 1,2,3,4,5);  
  PUT n1=;  
  n2 = WHICHN(4, 1,.,2,3,4,5);  
  PUT n2=;  
  n3 = WHICHN(4, 1,2,3,4,5);  
  PUT n3=;  
RUN;
```

This produces the following output:

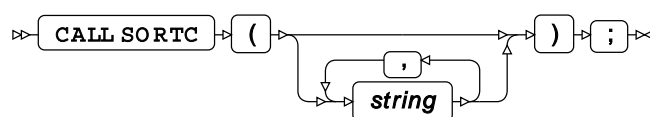
```
n1=0  
n2=5  
n3=4
```

The first example returns 0 because value 4.1 is not found in the value list.

The second example returns a different value than the third one because the missing value in the list is counted along with non-missing values.

CALL SORTC

Sorts a list of strings.



Sorts a list of strings, presented as arguments, into alphabetic order. The contents of the arguments are changed to match the sort order.

string

Optional argument

Type: Character

An argument containing a string to be sorted.

Basic example

In this example, the function is used to sort a list of strings. The result is written to the log.

```
DATA _NULL_;
  LENGTH c1 c2 c3 $ 11;
  c1 = 'Magnificent';
  c2 = 'London';
  c3 = 'Bikes';
  CALL SORTC(c1,c2,c3);
  PUT c1 " " c2 " " c3;
RUN;
```

This produces the following output:

```
Bikes  London  Magnificent
```

Example – sorting array of strings

In this example, the function is used to sort the strings in an array. The result is written to the log.

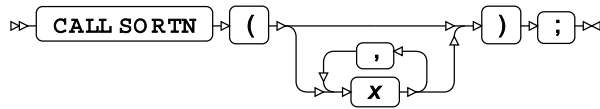
```
DATA _NULL_;
  ARRAY jj(12) $10 ('jack'    'jill'
                   'janet'   'john'
                   'humpty'  'dumpty'
                   'tom'     'jerry'
                   'captain' 'pugwash'
                   'black'   'pig');
  CALL SORTC(of jj(*));
  PUT jj(*);
RUN;
```

This produces the following output:

```
black captain dumpty humpty jack janet jerry jill john pig pugwash tom
```

CALL SORTN

Returns the values in a list of numeric variables sorted in ascending order.



Defined for numeric values only. Numeric and missing values in the variables given in the argument list are sorted in ascending order and re-assigned to these variables. Missing values are considered smaller than any non-missing values.

x

Optional argument

Type: Numeric

The point at which to calculate the values in a list of numeric variables sorted in ascending order.

Examples

In these examples, the values in a list of numeric variables sorted in ascending order is returned. The results are written to the log.

```
DATA _NULL_;  
  s1=0; s2=-1.1; s3=6.5; s4="4";  
  CALL SORTN (s1,s2,s3,s4);  
  
  PUT s1=;  
  
  PUT s2=;  
  
  PUT s3=;  
  
  PUT s4=;  
  
  m1=0; m2=-1.1; m3=-6.5; m4="four";  
  CALL SORTN (m1,m2,m3,m4);  
  
  PUT m1=;  
  
  PUT m2=;  
  
  PUT m3=;  
  
  PUT m4=;  
RUN;
```

This produces the following output:

```
s1=-1.1
s2=0
s3=4
s4=6.5

m1=.
m2=-1.1
m3=0
m4=6.5
```

The argument value "4" has been converted into a number, but the argument value "four" is considered missing. The sort order is adjusted accordingly, starting with the missing value.

String functions and CALL routines

Manipulate strings and characters. Strings are sequences of one or more characters.

You can find and replace characters and substrings, find the length of strings and substrings, and so on. Functions are also provided that enable you to manipulate strings and characters comprised of double-byte character set (DBCS) characters. A double-byte character consists of a lead byte and a trail byte and requires two consecutive storage bytes; it must be treated as a single unit in any operation involving characters and strings. The DBCS functions begin with a K (for example, KCOMPARE, KLENGTH and so on).

The string functions and CALL routines are grouped by function.

Calculate edit distances ↗	1942
Calculate the edit distances between strings.	
Change character case in strings ↗	1949
Change the case of strings.	
Compare strings ↗	1956
Compare strings in various ways.	
Concatenate strings ↗	1967
Concatenate strings in various ways, including or removing separators and spaces as required.	
Count characters or strings in a source string ↗	1978
Returns the number of specified characters or strings in source strings.	
Extract a substring from a source string ↗	1988
Returns a specified substring extracted from a source string.	
Find first character of a type ↗	2013
Return the first character of specified type in a source string.	
Find characters or rank in collating sequence ↗	2044
Return characters from specified positions in a collating sequence, or return the rank of a character in a sequence.	

Find position and length of substrings ↗	2046
Return the length and/or position of specified substrings in source strings.	
Modifying strings, characters and numerics ↗	2072
Return a source string modified in a particular way.	
Name literal check and manipulation ↗	2099
Check whether a string is a valid variable name, and convert strings to name literals.	
Remove spaces from a source string ↗	2102
Removes spaces from a specified string.	

Calculate edit distances

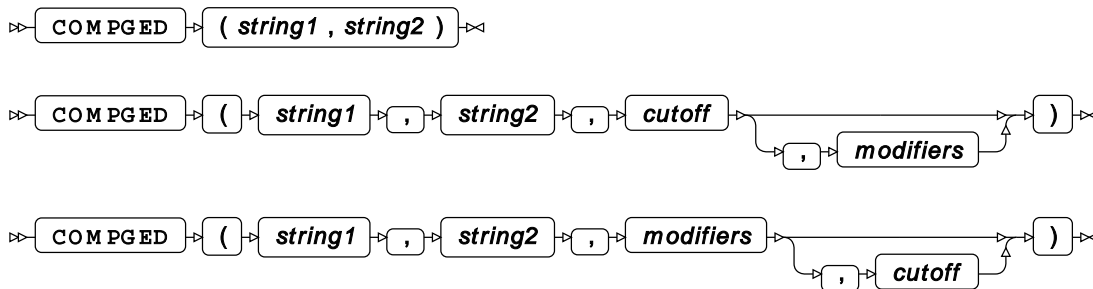
Calculate the edit distances between strings.

Edit distances can be calculated using various methods.

COMPGED ↗	1942
Returns the generalised edit distance between two strings.	
COMPLEV ↗	1945
Returns the Levenshtein edit distance between two strings.	
SPEDIS ↗	1946
Returns the spelling distance between two words.	
CALL COMPCOST ↗	1947
Adjusts the costs of various operations associated with the transformation of one string into another, such as deleting a character, or swapping one character with another.	

COMPGED

Returns the generalised edit distance between two strings.



Compares two strings and calculates an overall cost, based on the generalised edit distance, of transforming a specified string to another specified string. It can be regarded as a measure of the difference between two character strings. The generalised edit distance defines additional transformations to the standard (Levenshtein) edit distance.

Return type: Numeric

string1

Type: Character

The string to be compared to *string2*.

string2

Type: Character

The string to be compared to *string1*.

cutoff

Type: Numeric

The maximum cost to be returned by the function; above this value, costs will be ignored. For example if this argument is set to 100, and the function calculates that the cost of transformation is 200, the function will return 100.

modifiers

Optional argument

Type: Character

One or more characters that specify how the output should be modified. The following modifiers are available:

D

Provide detailed information on how the edit distance is calculated. The information is provided in a tabular format. See the example below.

I

Ignore case in the source string.

L

Ignore leading spaces in *string1*.

N

If the string is a name literal, ignore differences in case. For example, in `compged("Ford"N, "FoRd"N, 'N')` the case of `r` and `R` would be ignored.

:

If *string2* is longer than *string1*, ignore characters in *string2* beyond the length of *string1*. For example, `COMPGED('London Bikes', 'London Bikes s', ':');` would return the value 0 (zero).

The order of the *cutoff* and *modifiers* arguments can be swapped; for example `COMPGED('London Bikes','London Bokes', 100, 'I')` and `COMPGED('London Bikes','London Bokes', 'I', 100)` are equivalent.

Basic example

In this example, the function is used to find the generalised edit distance between two strings. The result is written to the log.

```
DATA _NULL_;
  result = COMPGED('London Bikes','London Bokes');
  PUT "The generalised edit distance is: " result;
RUN;
```

This produces the following output:

```
The generalised edit distance is: 100
```

Example – showing detailed informatin

In this example, the function is used to find the generalised edit distance between two strings. The `D` modifier is specified to provide detailed information. The result is written to the log.

```
DATA _NULL_;
  result = COMPGED('London Bikes','London Bokes',200,'D');
  PUT "The generalised edit distance is: " result;
RUN;
```

This produces the following output:

	L	o	n	d	o	n	B	i	k	e	s
	0	200	200	200	200	200	200	200	200	200	200
L	200	0	100	200	300	300	300	300	300	300	300
o	200	200	0	100	200	220	320	330	400	400	400
n	200	300	100	0	100	200	220	230	330	430	500
d	200	300	200	100	0	100	200	210	310	410	500
o	200	300	300	200	100	0	100	110	210	310	410
n	200	300	400	300	200	100	0	10	110	210	310
	200	300	400	310	210	110	10	0	100	200	300
B	200	300	400	410	310	210	110	100	0	100	200
o	200	300	300	400	410	310	210	200	100	100	200
k	200	300	400	400	500	410	310	300	200	200	100
e	200	300	400	500	500	510	410	400	300	300	200
s	200	300	400	500	600	600	510	500	400	400	300

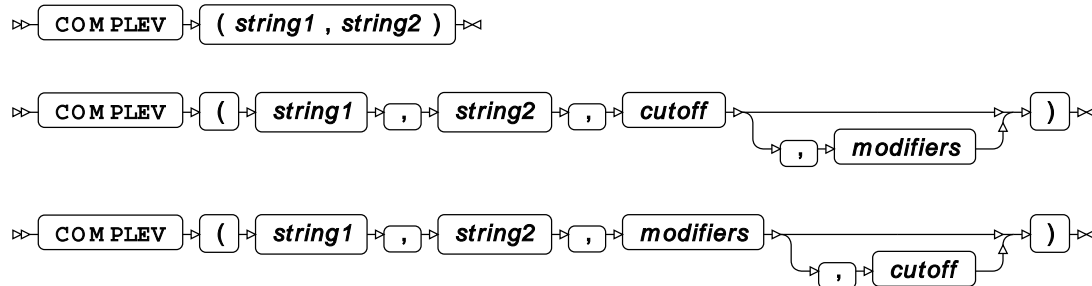
```

L      : MATCH
o      : MATCH
n      : MATCH
d      : MATCH
o      : MATCH
n      : MATCH
       : MATCH
B      : MATCH
o -> i : REPLACE
k      : MATCH
e      : MATCH
s      : MATCH
```

The generalised edit distance is: 100

COMPLEV

Returns the Levenshtein edit distance between two strings.



Compares two strings and calculates an overall cost, based on the Levenshtein edit distance, of transforming a specified string to another specified string. It can be regarded as a measure of the difference between two character strings.

Return type: Numeric

string1

Type: Character

The string to be compared to *string2*.

string2

Type: Character

The string to be compared to *string1*.

cutoff

Type: Numeric

The maximum cost to be returned by the function; above this value, costs will be ignored. For example if this argument is set to 10, and the function calculates that the cost of transformation is 12, the function will return 10. By default, there is no cut-off.

modifiers

Optional argument

Type: Character

Parameters that modify the output. These are optional. The following modifiers are available:

I

Ignore case in *string1*.

L

Ignore leading spaces in *string1*.

N

If the string is a name literal, ignore differences in case. For example, in `COMPLEV("Ford"N, "FoRd"N, 'N')` the case of r and R would be ignored.

:

If *string2* is longer than *string1*, ignore characters in *string2* beyond the length of *string1*. For example, `COMPLEV('London Bikes', 'London Bikes s', ':')` would return the value 0 (zero).

Example

In this example, the function is used to find the Levenshtein edit distance between two strings. This produces the following output:

```
DATA _NULL_;  
    result = COMPLEV('London Bikes', 'London Bokes');  
    PUT "The Levenshtein edit distance is: " result;  
RUN;
```

This produces the following output:

```
The Levenshtein edit distance is: 1
```

SPEDIS

Returns the spelling distance between two words.

» **SPEDIS** » (*string1* , *string2*) »

Determines the degree to which two words match, based on the spelling distance between them. The function returns the distance as an asymmetric value; that is, changing a to z is not the same as changing z to a.

Return type: Numeric

string1

Type: Character

The source string.

string2

Type: Character

The string to which *string1* is compared.

The spelling distance is calculated based on the transformations that would be required to make *string2* match *string1*. The following operations will be tried:

- Insert a letter
- Delete a letter
- Replace a letter
- Append a letter
- Delete the first letter
- Insert a first a letter
- Replace the first letter

Each operation has an associated cost. The function used default costs, but you can specify alternative costs using the `CALL COMPCOST` [↗](#) (page 1947) routine.

Example

In this example, the function is used to calculate the spelling distance between the target string `London` and the source string `Londin`. The result is written to the log.

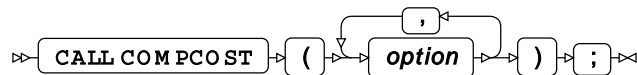
```
DATA _NULL_;  
  result = SPEDIS('London','Londin');  
  PUT 'The spelling distance is: ' result;  
RUN;
```

This produces the following output:

```
The spelling distance is: 16
```

CALL COMPCOST

Adjusts the costs of various operations associated with the transformation of one string into another, such as deleting a character, or swapping one character with another.



This `CALL` routine is most often called before the `COMPGED` function, which calculates the generalised edit distance for the transformation of two strings. The costs provided by `CALL COMPCOST` can be used to adjust the cost calculations performed by `COMPGED`.

option

Type: Character or numeric value

Specifies an editing operation, and the cost associated with that operation. The option is specified as a pair of values, a character operation name, enclosed in quotation marks, and a numeric cost for the operation. The format is:

```
'operation=', n
```

where *operation* is the name for an operation such as appending a character or inserting a blank. The following operations are available:

APPEND

Specifies the cost of appending a character.

BLANK

Specifies the cost of replacing a character with a blank.

DELETE

Specifies the cost of deleting a character.

DOUBLE

Specifies the cost of doubling a character (modifying o to be oo, for example).

FDELETE

Specifies the cost of deleting the first character in the string.

FINSERT

Specifies the cost of inserting a character at the first position in the string.

FREPLACE

Specifies the cost of replacing the first character in the string.

INSERT

Specifies the cost of inserting a character.

MATCH

Specifies the cost of matching a character.

PUNCTUATION

Specifies the cost of inserting a punctuation character.

REPLACE

Specifies the cost of replacing a character.

SINGLE

Specifies the cost of making a doubled character single (modifying oo to be o, for example).

SWAP

Specifies the cost of swapping one character with another.

TRUNCATE

Specifies the cost of truncating a string.

Example

In this example, `COMPGED` is first used to calculate the generalised edit distance between two strings based on the default costs used to calculate that distance. `CALL COMPCOST` is used to change the cost of inserting a value; `COMPGED` is then used to calculate the generalised edit distance using the new cost for inserting a character. The result is written to the log.

```
DATA _NULL_;
  result1 = COMPGED("London Bike","London Boke");
  PUT "Generalised edit distance is: " result1;
  call COMPCOST('replace=',10,'insert=',30);
  result2 = COMPGED("London Bike","London Boke");
  PUT "Generalised edit distance is now: " result2;
RUN;
```

This produces the following output:

```
Generalised edit distance is: 100
Generalised edit distance is now: 10
```

The generalised edit distance is first calculated to be 100. After executing `CALL COMPCOST`, the generalised edit distance is calculated to be 10.

Change character case in strings

Change the case of strings.

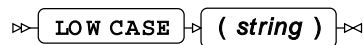
The case in strings can be changed to all upper, to all lower, and to sentence case.

<code>LOWCASE</code> ↗	1950
Returns a string in which all upper-case characters in a specified string have been converted to lower case.	
<code>KLOWCASE</code> ↗	1950
Returns a string in which all uppercase characters in a specified string of DBCS characters have been converted to lowercase.	
<code>KPROPCASE</code> ↗	1951
Returns a string in which the characters of a specified string of DBCS characters have been converted to another form, such as half-katakana to full-katakana or uppercase to lowercase.	
<code>KUPCASE</code> ↗	1953
Returns the string that results from converting all lowercase characters in a specified DBCS string to uppercase.	

PROPCASE ↗	1954
Returns the string that results from changing the case of words in a specified string so that the first letter is uppercase and other letters lowercase.	
UPCASE ↗	1955
Returns a string in which all lowercase characters have been converted to uppercase.	

LOWCASE

Returns a string in which all upper-case characters in a specified string have been converted to lower case.



Return type: Character

string

Type: Character

The string in which upper-case characters are to be converted.

Example

In this example, the function is used to examine the string for upper-case characters, which are then converted to lower-case characters. The result is written to the log.

```

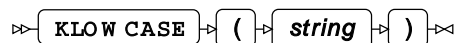
DATA _NULL_;
  result = lowercase('Magnificent Bike Company Bike A 1 2 50   London');
  PUT 'The converted string is: ' result;
RUN;
  
```

This produces the following output:

```
The converted string is: magnificent bike company bike a 1 2 50   london
```

KLOWCASE

Returns a string in which all uppercase characters in a specified string of DBCS characters have been converted to lowercase.



Converts all uppercase alphabetic characters in a string consisting of characters from a double-byte character set (DBCS) to corresponding lowercase characters, and returns the result. Strings in languages that have no equivalent lowercase characters are returned unmodified.

Return type: Character

string

Type: Character

A string in which uppercase characters are to be converted.

Example

In this example, the function is used to find upper case characters in the source strings; these are then converted to lowercase characters. The result is written to the log.

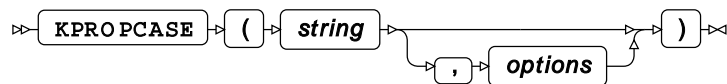
```
DATA _NULL_;  
    result = KLOWCASE('ΙΑΝΟΥΑΡΙΣ ΦΕΒΡΟΥΑΡΙΟΣ');  
    PUT result;  
RUN;
```

This produces the following output:

```
ιανουάρης φεβρουάριος
```

KPROPCASE

Returns a string in which the characters of a specified string of DBCS characters have been converted to another form, such as half-katakana to full-katakana or uppercase to lowercase.



Converts characters from a double-byte character set from one form to another, such as a half-width katakana character to a full-width katakana, or a lowercase character to an uppercase character, and returns the results.

Return type: Character

string

Type: Character

The string to be converted.

options

Optional argument

Type: Character

Options that define the type of conversion. The following are available:

HALF-KATAKANA, FULL-KATAKANA	Convert half-width katakana to full-width katakana.
------------------------------	---

FULL-KATAKANA, HALF-KATAKANA	Convert full-width katakana to half-width katakana.
KATAKANA, ROMAJI	Convert katakana characters to romaji characters.
ROMAJI, KATAKANA	Convert romaji characters to katakana characters.
HALF-ALPHABET, FULL-ALPHABET	Convert half-width characters (as used to display Korean or Chinese characters, for example) into full-width characters.
FULL-ALPHABET, HALF-ALPHABET	Convert full-width characters (as used to display Korean or Chinese, for example,) into half-width characters.
LOWERCASE, UPPERCASE	Convert lowercase characters to uppercase characters.
UPPERCASE, LOWERCASE	Convert uppercase characters to lowercase characters.

Convert full-width katakana to half-width

The following example changes Japanese katakana characters to half-katakana characters and back again.

```
DATA example;
  result1 = KPROPCASE('テレビ コンピューター', 'FULL-KATAKANA, HALF-KATAKANA');
  result2 = KPROPCASE(result1, 'HALF-KATAKANA, FULL-KATAKANA');
  PUT result1;
RUN;
```

This produces the following output:

```
テレビ コンピュー
テレビ コンピュー
```

Example – katakana characters to romaji and back

In this example, Japanese characters are changed to romaji characters, then changed back again.

```
DATA example;
  result = QUOTE(KRIGHT(text));
  result2 = CAT('熨斗目花色', KRIGHT(text));
  PUT result;
  PUT '熨斗目花色' result;
  PUT result2;
RUN;
```

This produces the following output:

```
TEREBI KONPYŪTĀ
テレビ コンピューター
```

Convert Uppercase Characters to Lowercase and Back

In this example, uppercase Greek characters are changed to lowercase characters, and then changed back again.

```
DATA _NULL_;  
    result1 = KPROPCASE('ΙΑΝΟΥΑΡΙΣ ΦΕΒΡΟΥΑΡΙΟΣ','uppercase, lowercase');  
    result2 = KPROPCASE(result1,'lowercase, uppercase');  
    PUT result1;  
    PUT result2;  
RUN;
```

This produces the following output:

```
λανουάρης φεβρουάριος  
ΙΑΝΟΥΑΡΙΣ ΦΕΒΡΟΥΑΡΙΟΣ
```

KUPCASE

Returns the string that results from converting all lowercase characters in a specified DBCS string to uppercase.

➤ **KUPCASE** ➤ (➤ *string* ➤) ➤

Converts all lowercase alphabetic characters in a string consisting of characters from a double-byte character set (DBCS) to corresponding uppercase characters, and returns the modified string. Strings in languages that have no equivalent uppercase characters are returned unmodified.

Return type: Character

string

Type: Character

A character or string of characters to be converted from lowercase to uppercase.

Example

In this example, the function is used to convert lowercase characters to uppercase. The result is written to the log.

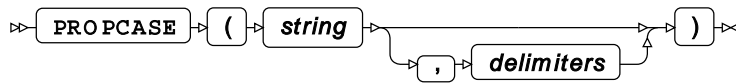
```
DATA _NULL_;  
    result = KUPCASE('λανουάρης φεβρουάριος');  
    PUT result;  
RUN;
```

This produces the following output:

```
ΙΑΝΟΥΑΡΙΣ ΦΕΒΡΟΥΑΡΙΟΣ
```

PROPCASE

Returns the string that results from changing the case of words in a specified string so that the first letter is uppercase and other letters lowercase.



Words in a string are separated by spaces by default, but a different separator can be used as a separator if required.

Return type: Character

string

Type: Character

A string containing words to be converted.

delimiters

Optional argument

Type: Character

One or more characters to be recognised as delimiters (separators).

Example – using default space separators

In this example, the function converts the case of words separated by spaces. The result is written to the log.

```
DATA _NULL_;
  result = PROPCASE('london bike company');
  PUT "Converted string is: " result;
RUN;
```

This produces the following output:

```
Converted string is: London Bike Company
```

Example – using a specified separator

In this example, the function converts the case of words separated by full-stops. The result is written to the log.

```
DATA _NULL_;
  result = PROPCASE('london bike.company', '.');
  PUT "Converted string is: " result;
RUN;
```


This produces the following output:

```
Converted string is: London bike.Company
```

Note:

If the string had been `london bike. company` (note the space after the full stop), the result would be `London bike. company`. In this example, the character after the full-stop is a space, rather than a letter, so there is nothing to convert.

Example – using multiple separators

In this example, the function converts the case of words separated by full-stops and commas. The result is written to the log.

```
DATA _NULL_;
  sentence='london,lovely bike.company';
  seps= ' , .';
  result = PROPCASE(sentence, seps);
  PUT "Converted string is: " result;
RUN;
```

This produces the following output:

```
Converted string is: London,Lovely bike.Company
```

UPCASE

Returns a string in which all lowercase characters have been converted to uppercase.

➤ **UPCASE** ➤ (*string*) ➤

Return type: Character

string

Type: Character

The string in which lowercase characters are to be converted.

Example

In this example, the function is used to convert lowercase characters to uppercase. The result is written to the log.

```
DATA _NULL_;
  result= UPCASE("Magnificent Bike Company A 1 2 50 London");
  PUT result;
RUN;
```

This produces the following output:

```
MAGNIFICENT BIKE COMPANY A 1 2 50 LONDON
```

Compare strings

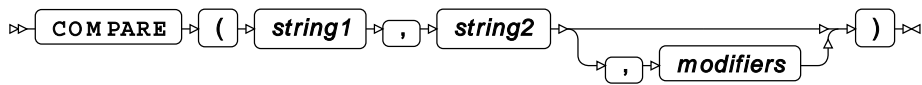
Compare strings in various ways.

Strings can be compared to see how alike they are, or to check which is higher or lower in the collating sequence.

COMPARE ↗	1956
Returns the first position at which two strings differ.	
KCOMPARE ↗	1958
Returns a value indicating whether two strings comprised of characters from a DBCS differ.	
LIKE ↗	1960
Returns a value indicating whether two strings are similar.	
MAXC ↗	1962
Returns the higher in the collating sequence of two strings after comparing them.	
MINC ↗	1964
Returns the lower in the collating sequence of two strings after comparing them.	
SOUNDSLIKE ↗	1966
Returns a value that indicates whether two strings are similar or not. The strings are converted to soundex equivalents, and then compared.	

COMPARE

Returns the first position at which two strings differ.



Return type: Numeric

string1

Type: Character

The string to be compared against *string2*.

string2

Type: Character

The string against which *string1* is compared.

modifiers

Optional argument

One or more modifiers. A modifier changes the operation of the function. The following modifiers are available:

"."

If *string2* is longer than *string1*, ignore characters in *string2* beyond the length of *string1*.

"I"

Differences in case when comparing alphabetic characters are ignored.

"L"

Ignore leading spaces in the comparison.

"N"

If the string is a name literal, differences in case are ignored. For example,

`COMPARE("Ford"N, "FoRd"N, 'N')` would return 0 (zero).

If the character that differs is higher in the collating sequence in *string1* than it is in *string2*, then the value returned is positive.

If the character that differs is higher in the collating sequence in *string2* than it is in *string1*, then the value returned is negative.

For example, comparing `Bicycle` to `Bacycle` returns 2, whereas comparing `Bacycle` to `Bicycle` returns -2.

Example – specifying no arguments

In this example, the function is used to compare two strings, with no arguments specified. The result is written to the log.

```
DATA _NULL_;  
    result = COMPARE("Ford", "Ford");  
    PUT "The result of the comparison is: " result;  
RUN;
```

This produces the following output:

```
The result of the comparison is: 0
```

The function returns 0 (zero), as both strings are identical.

Example – how collation sequence affects result

In this example, the function is used to compare two strings, with no arguments specified. The result is written to the log.

```
DATA _NULL_;  
  result = COMPARE("Ford", "Fprd");  
  PUT "The result of the comparison is: " result;  
RUN;
```

This produces the following output:

```
The result of the comparison is: -2
```

The function returns -2, which is the position of the first character that is different in the second string; the value is negative because *o* is lower in the collating sequence than *p*.

Example – using modifiers

In this example, the function is used to compare two strings while ignoring differences in case. The result is written to the log.

```
DATA _NULL_;  
  result = COMPARE("Ford", "FOrd", 'I');  
  PUT "The result of the comparison is: " result;  
RUN;
```

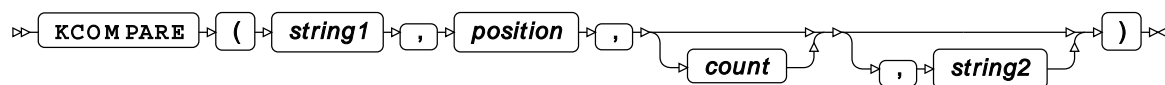
This produces the following output:

```
The result of the comparison is: 0
```

The function returns 0 (zero). Because *I* has been specified, the two strings are seen as identical.

KCOMPARE

Returns a value indicating whether two strings comprised of characters from a DBCS differ.



Compares two strings containing characters from a double-byte character set (DBCS), and returns a value that indicates whether they differ. You can compare the entire strings, or a substring to a string.

Return type: Numeric

string1

Type: Character

The source string to be compared.

position

Type: Character or numeric value

The position in *source* at which to start the comparison. If this argument is specified, *count* must also be specified.

count

Optional argument

Type: Character or numeric value

The number of characters to be compared. If this argument is specified, *position* must also be specified.

string2

Optional argument

Type: Character

The string to be compared against *source*.

If *position* and *count* are not specified, the entire string in *source* is compared to the entire string in *compare*. If the two strings do not differ, 0 (zero) is returned.

If *position* and *count* are specified, the substring in *source* is compared to the string in *compare*. If the two strings do not differ, 0 (zero) is returned.

If the strings or the substrings do differ, the following values are returned:

- 1 The first character that differs is lower in the collating sequence in *source* than it is in *compare*. For example, `kcompare('r', 's')` will return -1.
- 1 The first character that differs is higher in the collating sequence in *source* than it is in *compare*. For example, `kcompare('s', 'r')` will return 1.

See the examples below for more information.

Basic example

In this example, the function is used to compare two strings. The result is written to the log.

```
DATA _NULL_;  
    result = KCOMPARE("熨斗目花色", "熨斗目花色");  
    PUT result;  
RUN;
```

This produces the following output:

```
0
```

Both strings are exactly the same. As *position* and *count* are not specified, the entire strings are compared.

Example – showing affect of collation sequence

In this example, the function is used to compare two strings. The result is written to the log.

```
DATA _NULL_;  
    result = KCOMPARE("熨斗目花色", "熨目目花色");  
    PUT result;  
RUN;
```

This produces the following output:

```
-1
```

The value will be negative because 斗 is lower in the collating sequence than 目.

Example – specifying substrings

In this example, the function is used to compare a substring of the source to another string. The result is written to the log.

```
DATA _NULL_;  
    result = KCOMPARE("熨斗目花色", 3, 2, "目花");  
    PUT result;  
RUN;
```

This produces the following output:

```
0
```

The specified substring of the source string matches the string being compared.

In this example, the function is again used to compare a substring of the source to another string. The result is written to the log.

```
DATA _NULL_;  
    result = KCOMPARE("熨斗目花色", 3, 2, "目色");  
    PUT result;  
RUN;
```

This produces the following output:

```
1
```

The specified substring of the source does not match the string being compared, and the first character that does not match in the source is higher in the collating sequence.

LIKE

Returns a value indicating whether two strings are similar.

➤ **LIKE** ➤ (*string1* , *string2*) ➤

Compares two strings, and if they exactly alike, or match based on specified wildcards, returns 1 (true); otherwise returns 0 (false). Characters of different case are regarded as different; for example, A is not the same as a.

Return type: Numeric

string1

Type: Character

A string to be compared.

string2

Type: Character

A string to compare with *string1*.

string2 can contain the wildcards:

%	Any number of characters.
_	Any single character.

For example:

- Fird, Foord, Fooord and Fard match F%rd
- Ford and Fird match F_rd
- Fooord does not match F_rd

Example – Strings are alike, one character varies

In this example, the function is used to compare two similar strings. The result is written to the log.

```
DATA _NULL_;
  a="Magnificent";
  b="Magnifice_t";
  same = LIKE(a,b);
  result = IFC(same, 'Yes', 'No');
  PUT 'Are the strings alike? ' result;
RUN;
```

This produces the following output:

```
Are the strings alike? Yes
```

Example – Strings are alike, various characters differ

In this example, the function is used to compare strings. The result is written to the log.

```
DATA _NULL_;
  a="Magnificent";
  b="Mag%ice_t";
  same = LIKE(a,b);
  result = IFC(same, 'Yes','No');
  PUT 'Are the strings alike? ' result;
RUN;
```

This produces the following output:

```
Are the strings alike? Yes
```

The function returns 0, therefore the strings are not identical.

Example – Strings differ

In this example, the function is used to compare strings. The result is written to the log.

```
DATA _NULL_;
  a="Magnificent";
  b="Ma%ificeNt";
  same = LIKE(a,b);
  result = IFC(same, 'Yes','No');
  PUT 'Are the strings alike? ' result;
RUN;
```

This produces the following output:

```
Are the strings alike? No
```

Although the `gn` is matched through the wildcard `%`, the `N` differs in case.

MAXC

Returns the higher in the collating sequence of two strings after comparing them.

➤ **MAXC** ➤ (*string1* , *string2*) ➤

The function uses the collating sequence on the device on which the function runs. If one string is longer than the other, the shorter string is padded with spaces before the comparison. Case is ignored.

Strings are compared character by character. The higher string in the collating sequence is that string in which the first character that differs from the other is higher in the sequence. For example:

<i>string1</i>	<i>string2</i>	Result
London	London	London
Lundun	Luudun	Luudon

<i>string1</i>	<i>string2</i>	Result
b21e	B20e	b21e

In the last example, b21e is higher, even though uppercase characters are higher in the collating sequence, because case is ignored.

If two strings are the same, or are the same apart from case (for example, Bike and BlkE), *string1* is returned.

Return type: Character

string1

Type: Character

The character or string to be compared with *string2*.

string2

Type: Character

The character or string to be compared with *string1*.

<i>string1</i>	<i>string2</i>	Result
Lundun	London	Lundun
Lundun	Luudun	Luudon
b21e	B20e	B21e

Example – comparing characters

In this example, the function is used to compare two characters. The result is written to the log.

```
DATA _NULL_;
  result1 = maxc('M', 'm');
  result2 = maxc('m', 'M');
  result3 = maxc('M', 'o');
  PUT 'The higher character in the collating sequence is ' result1;
  PUT 'The higher character in the collating sequence is ' result2;
  PUT 'The higher character in the collating sequence is ' result3;
RUN;
```

This produces the following output:

```
The higher character in the collating sequence is M
The higher character in the collating sequence is m
The higher character in the collating sequence is o
```

In the first two uses of the function the characters are the same apart from case, so the first argument is returned: In the third use of the function, M is higher in the collating sequence than o, so M is returned.

Example – comparing strings

In this example, the function is used to compare two strings. The result is written to the log.

```
DATA _NULL_;  
  result = maxc('London  ', 'Londun');  
  PUT 'The higher string in the collating sequence is ' result;  
RUN;
```

This produces the following output:

```
The higher string in the collating sequence is Londun
```

The second string is padded with spaces to match the length of the first string. The function returns the value `Londun`, as `u` in the second string is the first character higher than `o` in the first.

MINC

Returns the lower in the collating sequence of two strings after comparing them.

➤ **MINC** ➤ (*string1* , *string2*) ➤

The function uses the collating sequence on the device on which the function runs. If one string is longer than the other, the shorter string is padded with spaces before the comparison. Case is ignored. If two strings are the same apart from case (for example, `Bike` and `BIkE`), the first argument is returned.

Strings are compared character by character. The lower string in the collating sequence is that string in which the first character that differs from the other is lower in the sequence. For example:

<i>string1</i>	<i>string2</i>	Result
London	London	London
Lundun	Luudun	Luudon
b21e	B20e	B20e

In the last example, `B20e` is lower, even though uppercase characters are higher in the collating sequence, because case is ignored.

If two strings are the same, or are the same apart from case (for example, `Bike` and `BIkE`), *string1* is returned.

string1

Type: Character

A character or string.

string2

Type: Character

A character or string.

Return type: Character

string1

Type: Character

A character or string.

string2

Type: Character

A character or string.

Example – comparing characters

In this example, the function is used to compare two characters. The result is written to the log.

```
DATA _NULL_;
  result1 = MINC('M', 'm');
  result2 = MINC('m', 'M');
  result3 = MINC('M', 'o');
  PUT 'The lower character in the collating sequence is ' result1;
  PUT 'The lower character in the collating sequence is ' result2;
  PUT 'The lower character in the collating sequence is ' result3;
RUN;
```

This produces the following output:

```
The lower character in the collating sequence is M
The lower character in the collating sequence is m
The lower character in the collating sequence is M
```

Example – comparing strings

In this example, the function is used to compare two strings. The result is written to the log.

```
DATA _NULL_;
  result = MINC('London   ', 'Londun');
  PUT 'The lower string in the collating sequence is ' result;
RUN;
```

This produces the following output:

```
The lower string in the collating sequence is London
```

The second string is padded with spaces to match the length of the first string. The function returns the value `London`, as `o` is the first character in the collating sequence lower than a matching character, `u` in the other string.

SOUNDSLIKE

Returns a value that indicates whether two strings are similar or not. The strings are converted to soundex equivalents, and then compared.

⇒ **SOUNDSLIKE** ⇒ (*string1* , *string2*) ⇒

If the strings are similar based on this comparison, 1 is returned; if they are not , 0 is returned.

Return type: Numeric

string1

Type: Character

The string to be compared with *string2*.

string2

Type: Character

The string to be against *string1*.

Example

In this example, the function is used to compare various strings. The result is written to the log.

```
DATA _NULL_;

    result1 = SOUNDSLIKE('Bike', 'Bike');
    result2 = SOUNDSLIKE('Bike', 'Bikes');
    result3 = SOUNDSLIKE('Bike', 'Buke');
    result4 = SOUNDSLIKE('Bike', 'Bkue');
    result5 = SOUNDSLIKE('Bike', 'Bite');

    rc = ifc(result1, "Bike and Bike do soundlike","Bike and Bike do not
soundlike" );
    PUT rc;
    rc = ifc(result1, "Bike and Bikes do soundlike","Bike and Bikes do not
soundlike" );
    PUT rc;
    rc = ifc(result1, "Bike and Buke do soundlike","Bike and Buke do not
soundlike" );
    PUT rc;
    rc = ifc(result1, "Bike and Bkue do soundlike","Bike and Bkue do not
soundlike" );
    PUT rc;
    rc = ifc(result1, "Bike and Bite do soundlike","Bike and Bite do not
soundlike" );
    PUT rc;

RUN;
```

This produces the following output:

```
Bike and Bike do soundlike
Bike and Bikes do not soundlike
Bike and Buke do soundlike
Bike and Bkue do soundlike
Bike and Bite do not soundlike
```

Because the comparison is based on soundex, `Bike` and `Bkue` are more similar than `Bike` and `Bite`.

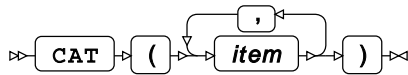
Concatenate strings

Concatenate strings in various ways, including or removing separators and spaces as required.

<code>CAT</code> ↗	1968
Returns the string that results from concatenating specified characters or strings. Spaces are preserved and included in the concatenated string.	
<code>CATQ</code> ↗	1968
Returns the string that results from concatenating specified characters or strings, adding or removing spaces and separators as specified using modifiers to the function.	
<code>CATS</code> ↗	1972
Returns the string that results from concatenating specified characters or strings after first stripping leading and trailing spaces.	
<code>CATT</code> ↗	1973
Returns the string that results from concatenating specified characters or strings after first stripping trailing spaces.	
<code>CATX</code> ↗	1974
Returns the string that results from concatenating specified characters or strings, where the strings and characters are separated by a specified separator.	
<code>CALL CATS</code> ↗	1975
Returns a string created by concatenating characters or strings, after stripping leading and trailing spaces from those characters or strings. The concatenated string is returned to an argument in the <code>CALL</code> routine.	
<code>CALL CATT</code> ↗	1975
Returns a string created by concatenating characters or strings, after stripping trailing spaces from those characters or strings. The concatenated string is returned to an argument in the <code>CALL</code> routine.	
<code>CALL CATX</code> ↗	1976
Returns a string created by concatenating characters or strings, inserting a specified separator between those characters or strings. The concatenated string is returned to a specified argument in the routine.	
<code>KSTRCAT</code> ↗	1977
Returns the string that results from concatenating specified DBCS characters or strings.	

CAT

Returns the string that results from concatenating specified characters or strings. Spaces are preserved and included in the concatenated string.



Return type: Character

item

Type: Character or numeric value

A character or string.

Example

In this example, the function concatenates the comma-separated list of strings to form a new string. The result is written to the log.

```
DATA _NULL_;
  result = CAT("    Magnificent"," Bicycle    "," Company");
  PUT "Concatenated string: " result;
RUN;
```

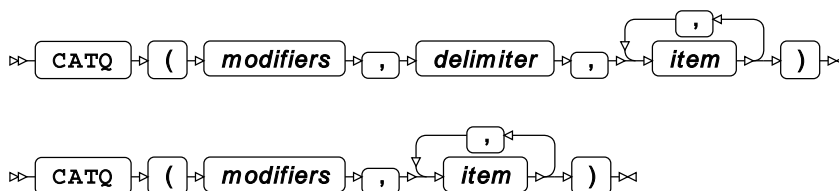
This produces the following output:

```
Concatenated string: Magnificent Bicycle    Company
```

Strings have been concatenated and their spaces preserved (except for the first string where leading spaces have been removed by the PUT operation). No separators have been inserted between strings.

CATQ

Returns the string that results from concatenating specified characters or strings, adding or removing spaces and separators as specified using modifiers to the function.



Provides a flexible way to concatenate characters or strings to create another string. You can specify which separators separate strings, and whether:

- Spaces will be stripped from strings

- Strings will be enclosed in quotation marks

Return type: Character

modifiers

One or more characters that specify how the output should be modified. You can specify more than one modifier. For example, you can specify 'C', or 'CA1S'.

The following modifiers are available:

"1"

Specifies that modifiers that insert quotation marks to delimit substrings in the concatenated string should use single quotation marks. By default, double quotation marks are used in such instances. You can alternatively specify the single quotation mark (') in place of 1.

You can use this modifier with the modifiers A and B; for example, you can specify A1 or A '. If you specify this modifier without also specifying the A and B modifier, results are unpredictable.

"2"

Specifies that modifiers that insert quotation marks to delimit substrings in the concatenated string should use double quotation marks. You can alternatively specify the double quotation mark (") in place of 2.

This modifier can be used with the A or B modifiers; for example, you can specify A2 or A".

If you specify this modifier without also specifying the A or B modifier, results are unpredictable.

"A"

Strings will be enclosed in quotation marks in the resulting string. Any leading and trailing spaces in the strings are preserved.

By default, double quotation marks will be used, but this modifier can be used with the 1 or 2 modifier, described above, to define the type of quotation mark to be used. For example, specifying the modifier A1 will enclose a string in single quotation marks.

"B"

Strings with leading and/or trailing spaces are enclosed in quotation marks in the resulting string, and separated by a space character.

By default, double quotation marks will be used, but this modifier can be used with the 1 or 2 modifier, described above, to define the type of quotation mark to be used. For example, specifying the modifier B1 will enclose a string in single quotation marks.

"C"

Strings are separated by commas in the resulting string. Any leading and trailing spaces in the strings are preserved.

"D"

Strings are separated by a specified delimiter in the resulting string.

If you specify this modifier, you must specify the delimiter; otherwise, the first string or character in your put is used as the delimiter. Any leading and trailing spaces in the strings are preserved. See *delimiter* below for more information.

"H"

Strings are separated by horizontal tabs in the resulting string. Any leading and trailing spaces in the strings are preserved.

"M"

Use a delimiter to separate all strings and characters, even nulls and hidden characters. This modifier should be used with the **D** modifier to specify a delimiter.

"N"

If the format of a resulting string is not a valid data variable name (for example, it starts with a numeric or contains spaces), it is converted to a name literal by wrapping the string in quotation marks and appending an **N** character.

"Q"

Retains the quotation marks around the strings, and maintains the leading and trailing spaces exactly as they exist between the quotation marks.

Strings are separated by spaces in the resulting string. If a string contains quotation marks, the quotation marks in the resulting string are set such that the result will not cause an error. For strings that do not contain quotation marks, the quotation mark for the string in the resulting string will be the default, or that set by the **A** modifier.

For example, the **CAT** function `cat(' "Hello"', '"World" ', ' Programming');` would return the concatenated string `"Hello" "World" Programming`. The quoted quotation marks and leading and trailing spaces have been retained, except for the leading spaces of the first string, which have been removed as expected for **CAT**. However, if you enter the same strings into **CATQ** with the **Q** modifier:

```
catQ('q', ' "Hello"', '"World" ', ' Programming')
```

The output would be as follows:

```
' "Hello" ' '"World" ' " Programming"
```


Here, the leading and trailing spaces have been preserved, and the style of the quotation marks used from the input strings to surround the strings in the concatenated string. However, the final string, " Programming" has been surrounded with double quotation marks, as there are no other quotation marks to preserve, and double quotation marks are the default marks for delimiting strings.

"S"

Strips leading and trailing spaces in strings in the resulting string, and separates the strings with spaces. Strings that contain spaces are wrapped in quotation marks.

"T"

Strips trailing spaces in strings in the resulting string, and separates the strings with spaces. If strings still contain spaces, they are wrapped in quotation marks.

"X"

Any non-printable character in the input is replaced by its hexadecimal equivalent.

delimiter

Type: Character

A character or string to be used as a separator for concatenated strings. *delimiter* must be specified with the `D` modifier.

item

Type: Character or numeric value

A character or string.

Unless the `Q` option is specified, strings are concatenated without the quotation marks used to define them as strings. However, if a string contains a space, the string is concatenated with its quotation marks. For example:

```
DATA _NULL_;  
  x = CATQ(, "Hello","World","Prog ramming");  
  PUT x;  
RUN;
```

This produces the following output:

```
Hello World "Progr amming"
```

Example – compressing spaces and inserting Separators using CTA Modifiers

In this example, a new string will be created by concatenating the strings in the function. Multiple modifiers are used to:

- Strip trailing spaces in the strings before concatenation (`T` modifier)

- Insert commas as separators items in the resulting string (C modifier)
- Wrap each concatenated string in single quotes, (A and 1 modifiers)

The result is written to the log.

```
DATA _NULL_;
  string1='Programming';
  mods='CA1T';
  result = catq(mods, '      Hello',' World      ', string1);
  PUT "Concatenated string: " result;
RUN;
```

This produces the following output:

```
Concatenated string: '      Hello',' World','Programming'
```

Example – compressing spaces and inserting separators using DS modifiers

In this example, a new string will be created by concatenating the strings in the function. Multiple modifiers have been used to:

- Strip all leading and trailing spaces in strings before concatenation (S modifier)
- Separate concatenated strings in the resulting string using the ** characters as specified by the D modifier and the second argument

The result is written to the log.

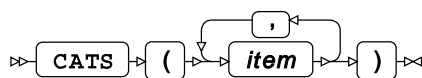
```
DATA _NULL_;
  string1='Programming';
  result = CATQ('DS', '**', '      Hello',' World      ', string1);
  PUT "Concatenated string: " result;
RUN;
```

This produces the following output:

```
Concatenated string: Hello**World**Programming
```

CATS

Returns the string that results from concatenating specified characters or strings after first stripping leading and trailing spaces.



Return type: Character

item

Type: Character or numeric value

A character or string.

Example

In this example, the function concatenates the comma-separated list of strings to form a new string, stripping leading and trailing spaces as it does so. The result is written to the log.

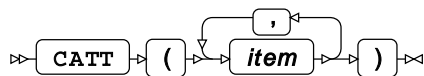
```
DATA _NULL_;  
    result = CATS("    Hello", " World    ", " Programming");  
    PUT "Concatenated string: " result;  
RUN;
```

This produces the following output:

```
Concatenated string: HelloWorldProgramming
```

CATT

Returns the string that results from concatenating specified characters or strings after first stripping trailing spaces.



Return type: Character

item

Type: Character or numeric value

A character or string.

Example

In this example, the function concatenates the comma-separated list of strings to form a new string, stripping the trailing spaces. The result is written to the log.

```
DATA _NULL_;  
    result = CATT("    Hello", "World    ", " Programming");  
    PUT "Concatenated string: " result;  
RUN;
```

This produces the following output:

```
Concatenated string: HelloWorld Programming
```

Whether leading spaces are removed from the resulting concatenated string depends on the operation on the result; for example, the `PUT` in this example removes them, but a `QUOTE` would retain them.

CATX

Returns the string that results from concatenating specified characters or strings, where the strings and characters are separated by a specified separator.



Return type: Character

separator

Type: Character

A character or string that will be used as the string separator.

item

Type: Character or numeric value

A character or string, or a variable containing a character or string.

Example

In this example, the function concatenates the comma-separated list of strings to form a new string, first stripping the leading and trailing spaces from each string and inserting the specified separator between them. This produces the following output:

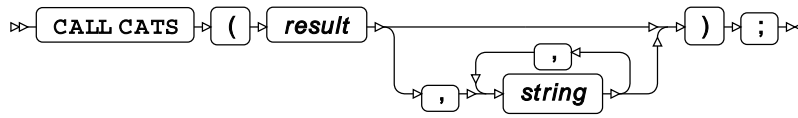
```
DATA _NULL_;  
    result = CATX('*', "    Hello","World    "," Programming");  
    PUT "Concatenated string: " result;  
RUN;
```

This produces the following output:

```
Concatenated string: Hello*World*Programming
```

CALL CATS

Returns a string created by concatenating characters or strings, after stripping leading and trailing spaces from those characters or strings. The concatenated string is returned to an argument in the `CALL` routine.



result

Type: Character

An argument into which the concatenated string is returned.

string

Optional argument

Type: Character or numeric value

A character or string to be concatenated.

Example

In this example, the routine concatenates the comma-separated list of strings to form a new string, stripping the leading and trailing spaces. The concatenated string is returned to the `strng` variable. The result is written to the log.

```

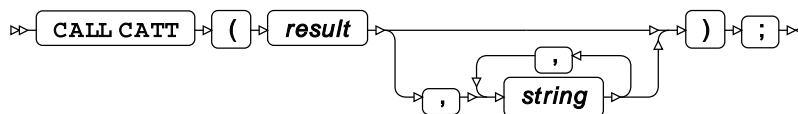
DATA _NULL_;
  LENGTH strng $ 17;
  CALL CATS(strng,'London','      Bike',' Company ');
  PUT "The concatenated string is " strng;
RUN;
  
```

This produces the following output:

```
The concatenated string is LondonBikeCompany
```

CALL CATT

Returns a string created by concatenating characters or strings, after stripping trailing spaces from those characters or strings. The concatenated string is returned to an argument in the `CALL` routine.



result**Type:** Character

An argument into which the concatenated string is returned.

string

Optional argument

Type: Character or numeric value

A character or string to be concatenated.

Example

In this example, the routine concatenates the comma-separated list of strings to form a new string, stripping the trailing spaces. The concatenated string is returned to the *strng* variable. The result is written to the log.

```
DATA _NULL_;  
  LENGTH strng $ 26;  
  call CATT(strng,"    Hello","World    "," Programming");  
  result = QUOTE(strng);  
  PUT "The concatenated string is " result;  
RUN;
```

This produces the following output:

```
The concatenated string is "    HelloWorld Programming"
```

Note:

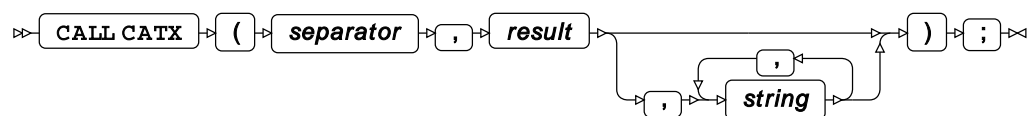
If PUT had been used to write *strng* to the log without quotes, the result would have been:

```
HelloWorld Programming
```

The leading spaces would have been removed.

CALL CATX

Returns a string created by concatenating characters or strings, inserting a specified separator between those characters or strings. The concatenated string is returned to a specified argument in the routine.

**separator****Type:** Character

A character or string that will be used as the string separator.

result

Type: Character

The argument into which the string is returned.

string

Optional argument

Type: Character or numeric value

A character or string.

Example

In this example, the routine strips the leading and trailing spaces from each comma-separated string, and then concatenates each string, inserting the specified separator between strings, to form a new string. The concatenated string is returned to the *strng* variable. The result is written to the log.

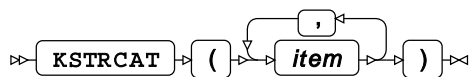
```
DATA _NULL_;  
  LENGTH strng $ 25;  
  CALL CATX('**', strng, " Hello      "," World "," Programming");  
  PUT "The concatenated string is " strng;  
RUN;
```

This produces the following output:

```
The concatenated string is Hello**World**Programming
```

KSTRCAT

Returns the string that results from concatenating specified DBCS characters or strings.



Concatenates a comma-separated list of double-byte character set (DBCS) characters, or strings consisting of such characters, to create another string. Any spaces in the strings are preserved and included in the concatenated string. Otherwise, the strings are concatenated without spaces.

Return type: Character

item

Type: Character

A character or string.

Example

In this example, the function concatenates the comma-separated list of strings to form a new string. The result is written to the log.

```
DATA _NULL_;
  result1 = KSTRCAT("青碧", "熨斗目花色", "青柳鼠");
  result2 = KSTRCAT(" 青碧", " 熨斗目花色  ", " 青柳鼠");
  PUT "The concatenated string is: " result1;
  PUT "The concatenated string is: " result2;
RUN;
```

This produces the following output:

```
The concatenated string is: 青碧熨斗目花色青柳鼠
The concatenated string is: 青碧 熨斗目花色 青柳鼠
```

In the first result, the strings have been concatenated. No separators have been inserted between strings.

In the second result, the strings have been concatenated and their spaces retained (except for the first string where leading spaces have been stripped by the `PUT`). No separators have been inserted between strings. The only spaces are those associated with the original strings.

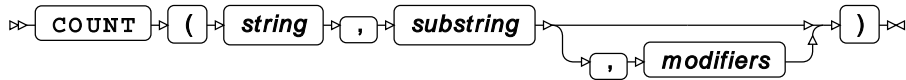
Count characters or strings in a source string

Returns the number of specified characters or strings in source strings.

COUNT 🔗	1978
Returns the number of occurrences of a specified string in a source string.	
COUNTC 🔗	1980
Returns the number of occurrences of specified characters in a source string.	
COUNTW 🔗	1982
Returns the number of words in a specified string. The character that separates words can be specified.	
KCOUNT 🔗	1988
Returns the number of characters in a string that require more than one byte to represent them in the current character set.	

COUNT

Returns the number of occurrences of a specified string in a source string.



Return type: Numeric

string

Type: Character

The string to be examined.

substring

Type: Character

The string to find in *source*.

modifiers

Optional argument

One or more characters that specify how the output should be modified. The following modifiers are available:

"I"

Ignore case in the source string.

"T"

Trim trailing spaces from both arguments.

"I"

Ignore case in the source string.

"T"

Trim trailing spaces from both arguments.

Example – finding a matching string

In this example, the function is used to find the number of times `Bike` is found in the source string. The result is written to the log.

```
DATA _NULL_;  
  search = 'Bike';  
  result = COUNT('Magnificent Bike Company (Bikes)', search);  
  PUT "The string is found " result "times";  
RUN;
```

This produces the following output:

```
The string is found 2 times
```

The string is found twice, so 2 is returned.

Example – finding matches by ignoring case

In this example, the function is used to match a string while ignoring case. The result is written to the log.

```
DATA _NULL_;  
  search = 'company';  
  result = COUNT('Magnificent Bike Company (Bikes)', search, 'I');  
  PUT "The string is found " result "times";  
RUN;
```

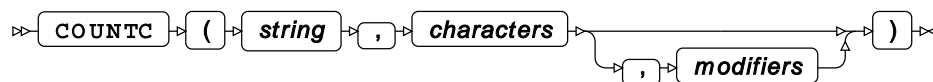
This produces the following output:

```
The string is found 1 times
```

The string `company` matches `Company` because the `I` modifier has been set, so 1 is returned.

COUNTC

Returns the number of occurrences of specified characters in a source string.



Return type: Numeric

string

Type: Character

The string to be examined.

characters

Type: Character

One or more characters to find in *source*.

modifiers

Optional argument

Parameters that modify the output. The following modifiers are available:

"I"

Ignore case in the source string.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"T"

Trim trailing spaces from both arguments.

"V"

Return the number of characters that do not match the search.

"I"

Ignore case in the source string.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"T"

Trim trailing spaces from both arguments.

"V"

Return the number of characters that do not match the search.

Example – searching for characters

In this example, the function is used to find the characters `i` and `B` in the source string. The result is written to the log.

```
DATA _NULL_;
  search = 'iB';
  result = COUNTC('Magnificent Bike Company Bike A 1 2 50 London', search);
  PUT "Specified character(s) found " result "times";
RUN;
```

This produces the following output:

```
Specified character(s) found 6 times
```

The character `B` is found twice in the two instances of `Bike`, and `i` four times in `Bike` and `Magnificent`.

Example – returning the count of unmatched characters

In this example, the function is used to find the character `i` in the source string. The modifier `V` specifies that the value returned is the number of characters that do *not* match the search string. The result is written to the log.

```
DATA _NULL_;
  search = 'i';
  result = COUNTC('Magnificent Bike Company Bike A 1 2 50 London', search, 'V');
  PUT "Other character(s) found " result "times";
RUN;
```

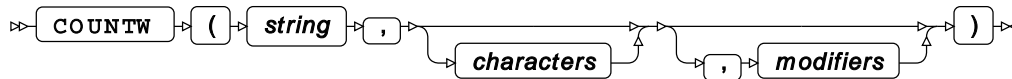
This produces the following output:

```
Other character(s) found 43 times
```

The character `i` is found four times; as there are 47 characters in the source string, 43 is returned.

COUNTW

Returns the number of words in a specified string. The character that separates words can be specified.



By default words are separated by spaces and punctuation marks. Multiple adjacent separators are treated as one separator.

Return type: Numeric

string

Type: Character

The string to be examined.

characters

Optional argument

Type: Character

One or more characters that identify the separator between words in *source*. By default, this function uses a space as a separator.

modifiers

Optional argument

One or more characters that specify how the output should be modified. For example, you might specify 'A', or 'AIM'.

The following modifiers are available:

"A"

Alphabetic characters will be used as separators.

"B"

Examine *source* backwards; that is, start searching from the right-hand side of the string towards the left-hand side.

"C"

Control characters will be used as separators.

"D"

Decimal numbers (0 through 9) will be used as separators.

"F"

A character that is a valid character for the first position in a variable name will be considered a separator.

"G"

Graphics characters will be used as separators.

"H"

Horizontal tabs will be used as separators.

"I"

If an alphabetic character is set as a separator, ignore the case of the character.

"K"

Causes modifiers to act in the opposite way. So, for example, if `L` is set, instead of removing all lower-case characters, the function will instead remove all uppercase characters.

"L"

Lower-case alphabetic characters will be used as separators.

"M"

Specifies that multiple adjacent separators delimit nulls. By default, multiple adjacent separators are assumed to operate as a single separator. In this example, this option is not specified (the default):

```
DATA _NULL_;  
    result = COUNTW('london,,bike,company;;A1', ',',';');  
    PUT result;  
RUN;
```

The function would return 4, as the adjacent separators are treated as a single separator. However, if you specified the `M` modifier:

```
DATA _NULL_;  
    result = COUNTW('london,,bike,company;;A1', ',',';', 'M');  
    PUT result;  
RUN;
```

the function would return 7, as adjacent separators are now treated as individual separators each delimiting a null.

"N"

A character that is a valid character for a variable name will be considered a separator.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"P"

Punctuation marks will be considered separators.

"Q"

Separators in strings delimited by quotation marks will be ignored. For example:

```
DATA _NULL_;  
    result1 = COUNTW("The London Bike Company" A1 London    ', ' ', 'Q');  
    result2 = COUNTW("The London Bike Company" A1 London    ', ' ');  
    PUT result1;  
    PUT result2;  
RUN;
```

In this example, the following is written to the log:

```
3  
6
```

In the first use of the function, the first word is `The London Bike Company`, as the `Q` modifier ensures the quoted text is counted as one word; the function therefore counts three words. In the second use of the function, spaces within the quoted string are identified as separators, as the `Q` modifier is not specified; the function therefore counts six words.

"R"

Trims the result.

"S"

White space (including spaces and tabs) is used as a separator.

"T"

Removes trailing spaces from each substring in the source string before applying separators.

"U"

Uppercase alphabetic characters will be used as separator.

"W"

Use print characters as separator.

"X"

Use all characters that can constitute hexadecimal numbers (0-9, A-F) as separator.

"A"

Alphabetic characters will be used as separators.

"B"

Examine *source* backwards; that is, start searching from the right-hand side of the string towards the left-hand side.

"C"

Control characters will be used as separators.

"D"

Decimal numbers (0 though 9) will be used as separators.

"F"

A character that is a valid character for the first position in a variable name will be considered a separator.

"G"

Graphics characters will be used as separators.

"H"

Horizontal tabs will be used as separators.

"I"

If an alphabetic character is set as a separator, ignore the case of the character.

"K"

Causes modifiers to act in the opposite way. So, for example, if `L` is set, instead of removing all lower-case characters, the function will instead remove all uppercase characters.

"L"

Lower-case alphabetic characters will be used as separators.

"M"

Specifies that multiple adjacent separators delimit nulls. By default, multiple adjacent separators are assumed to operate as a single separator. In this example, this option is not specified (the default):

```
DATA _NULL_;  
    result = COUNTW('london,,bike,company;;A1', ' ', ';');  
    PUT result;  
RUN;
```

The function would return 4, as the adjacent separators are treated as a single separator. However, if you specified the M modifier:

```
DATA _NULL_;  
    result = COUNTW('london,,bike,company;;A1', ' ',';', 'M');  
    PUT result;  
RUN;
```

the function would return 7, as adjacent separators are now treated as individual separators each delimiting a null.

"N"

A character that is a valid character for a variable name will be considered a separator.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"P"

Punctuation marks will be considered separators.

"Q"

Separators in strings delimited by quotation marks will be ignored. For example:

```
DATA _NULL_;  
    result1 = COUNTW('"The London Bike Company" A1 London', ' ',';', 'Q');  
    result2 = COUNTW('"The London Bike Company" A1 London', ' ',' ');  
    PUT result1;  
    PUT result2;  
RUN;
```

In this example, the following is written to the log:

```
3  
6
```

In the first use of the function, the first word is `The London Bike Company`, as the Q modifier ensures the quoted text is counted as one word; the function therefore counts three words. In the second use of the function, spaces within the quoted string are identified as separators, as the Q modifier is not specified; the function therefore counts six words.

"R"

Trims the result.

"S"

White space (including spaces and tabs) is used as a separator.

"T"

Removes trailing spaces from each substring in the source string before applying separators.

"U"

Uppercase alphabetic characters will be used as separator.

"W"

Use print characters as separator.

"X"

Use all characters that can constitute hexadecimal numbers (0-9, A-F) as separator.

Example – unmodified count

In this example, the function counts the words in the source string with no modifiers applied, so the words are by default separated by spaces and punctuation characters. The result is written to the log.

```
DATA _NULL_;  
  result = COUNTW('Magnificent Bike Company. A 1 2 50-London');  
  PUT result "words were found ";  
RUN;
```

This produces the following output:

```
8 words were found
```

Example – using specified separator

In this example, the function counts the words in the source string with a separator applied; in this case, the space character. The result is written to the log.

```
DATA _NULL_;  
  search = 'i';  
  result = COUNTW('Magnificent Bike Company. A 1 2 50-London', ' ');  
  PUT result "words were found ";  
RUN;
```

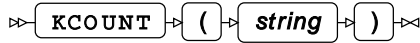
This produces the following output:

```
7 words were found
```

As the separator has been expressly set to the space character, `Company.` and `50-London` are each a single string; the source string therefore contains seven words.

KCOUNT

Returns the number of characters in a string that require more than one byte to represent them in the current character set.



Counts the number of characters in a string that require more than one byte to represent them in the character set used during the current session, and returns that count.

Return type: Numeric

string

Type: Character

The string to be examined for DBCS characters.

Example

In this example, the function is used to count the characters requiring more than one byte to represent them in a string containing both Latin and Japanese characters. The locale of WPS Workbench would need to be set to `Japanese_Japan` to enable the example to work correctly. The result is written to the log.

```

DATA _NULL_;
  result = KCOUNT('SIN(x) の冪級数展開. その2: 奇数{1,3,5,7}');
  PUT result;
RUN;

```

This produces the following output:

```
11
```

Extract a substring from a source string

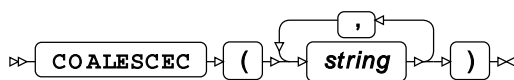
Returns a specified substring extracted from a source string.

COALESCEC ↗	1989
Returns the first string that contains characters other than all spaces or null.	
KSCAN ↗	1990
Returns the word at a specified index position in a string consisting of DBCS characters.	
KSUBSTR ↗	1991
Returns a substring from a source string consisting of DBCS characters, starting at a specified character.	

KSUBSTRB ↗	1993
Returns a substring from a source string consisting of DBCS characters, starting at a specified byte.	
SCAN ↗	1994
Returns the word at a specified index position in a string. The separator between words can be specified, if required. Various modifiers can be also be used to specify additional delimiters and to modify the way the source string is searched.	
SCANQ ↗	1998
Returns the word at a specified index position in a string. The separator between words can be specified if required.	
SUBPAD ↗	2000
Returns the substring of a specified length that starts at a specified position in a source string, and pads the returned substring to a specified length if required.	
SUBSTR ↗	2001
Returns either a substring of a specified length from a source string, or the string that results from replacement of a specified substring in a source string.	
SUBSTRN ↗	2004
Returns a substring of a specified length from a specified starting point in a source string.	
CALL SCAN ↗	2006
Returns the position and/or length of the substring at the specified index position in a source string.	
CALL SCANQ ↗	2011
Returns the position and/or length of the substring at the specified index position in a source string, specifying the separator between substrings.	

COALESCEC

Returns the first string that contains characters other than all spaces or null.



Return type: Character

string

Type: Character

A string to be examined.

Example

In this example, the function searches the strings provided to it for the first that does not contain only spaces or is null. The result is written to the log.

```
DATA _NULL_;  
  empty="  ";  
  result = COALESCEC(' ', ' ', empty, ' World Programming');  
  PUT "The resulting string is: " result;  
RUN;
```

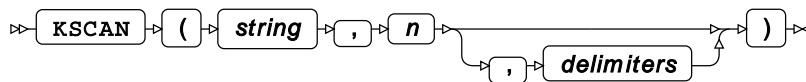
This produces the following output:

```
The resulting string is: World Programming
```

The first string provided to the function is null, the second string is all spaces, and the variable `empty` also contains all spaces; the function therefore returns `World Programming`.

KSCAN

Returns the word at a specified index position in a string consisting of DBCS characters.



Finds and returns a specified index position in a source string containing characters from a double-byte character set (DBCS).

The source string is treated as a list of words delimited by spaces (by default) or any other character you specify. Each word occupies a position in the list from 1–*n* and can be accessed using this function.

Return type: Character

string

Type: Character

The string to be examined.

n

Type: Numeric

The position of the word in the string you want to examine. This argument is mandatory.

delimiters

Optional argument

Type: Character

A delimiter used to separate words. You can specify more than one delimiter.

If you do not specify a delimiter, spaces (but not tabs) are recognised as delimiters. If you specify delimiters, spaces are still recognised as delimiters. Tabs are not recognised as spaces, so if the source includes tab delimiters you must enter a tab in *delimiter*.

Basic example

In this example, the function is used to find the fourth word in a string. The result is written to the log. No delimiter is specified, so spaces are used as delimiters.

```
DATA _NULL_;
  result = KSCAN('青碧 熨斗目花色 ときがら茶 猩々緋 青柳鼠',4);
  PUT "The fourth word is: " result;
RUN;
```

This produces the following output:

```
The fourth word is: 猩々緋
```

Example – source using specified delimiters

In this example, the function is used to find the fourth word in a string. The result is written to the log.

```
DATA NULL;
  word1 = KSCAN('青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶',1,', ');
  word2 = KSCAN('青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶',2,', ');
  word3 = KSCAN('青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶',3,', ');

  PUT "The first word is: " word1;
  PUT "The second word is: " word2;
  PUT "The third word is: " word3;
RUN;
```

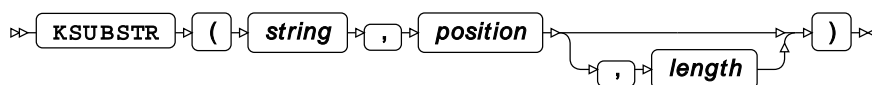
This produces the following output:

```
The first word is: 青碧 熨斗目花色
The second word is: 猩々緋 青柳鼠
The third word is: ときがら茶
```

The first delimiter found is 、 so the first two strings are seen as one word. The second delimiter found is a tab (the space in 猩々緋 青柳鼠 is ignored), so the third and fourth strings are seen as one word. The third word ときがら茶 appears after the second delimiter, the tab.

KSUBSTR

Returns a substring from a source string consisting of DBCS characters, starting at a specified character.



Finds and returns the substring that starts at a specified position in a source string consisting of double-byte character set (DBCS) characters. You can also optionally specify the length of string to return.

Return type: Character

string

Type: Character

The source string.

position

Type: Numeric

The position in the source string at which to return the substring of *length*.

length

Optional argument

Type: Numeric

The number of characters of the source string to be read.

If you do not specify *length*, the substring found starting at *position* through to the end of the source string is returned. If *length* would result in an attempt to read over the end of *string*, the length is assumed to be through to the end of the *string*. If *length* is set to 0 (zero), no substring is returned.

Basic example

In this example, the function is used to find and return the string that begins at position 18 and is six characters long. The result is written to the log.

```
DATA _NULL_;  
  result=QUOTE(KSUBSTR("青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶", 18,6));  
  PUT "The substring found is: " result;  
RUN;
```

This produces the following output:

```
The substring found is: 柳鼠 ときが
```

Example – reading over the end of string

In this example, the function is used to find and return the string that begins at position eighteen and is fifteen characters long. The result is written to the log.

```
DATA _NULL_;  
  result=KSUBSTR("青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶", 18,15);  
  PUT "The substring found is: " result;  
RUN;
```

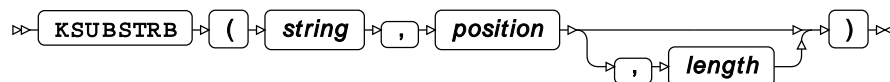
This produces the following output:

```
The substring found is: 柳鼠 ときから茶
```

In this example, there are fewer than fifteen characters after character position eighteen, so the substring from position eighteen through to the end of the string is returned.

KSUBSTRB

Returns a substring from a source string consisting of DBCS characters, starting at a specified byte.



Finds and returns the substring that starts at a specified byte in a source string comprised of double-byte character set characters. You can also optionally specify the length of string to return.

Return type: Character

string

Type: Character

The source string.

position

Type: Numeric

The byte position in the source string at which to return the substring.

length

Optional argument

Type: Numeric

The number of bytes of the source string to be read.

If you do not specify *length*, the substring found starting at *position* through to the end of the source string is returned. If *length* would result in an attempt to read over the end of the string, an error message is returned with the result; the result is the substring from the starting position specified through to the end of the string. If *length* is set to 0 (zero), no substring is returned.

Basic example

In this example, the function is used to find and return the string that begins at byte eighteen and is 24 bytes long. The result is written to the log.

```
DATA _NULL_;  
    result=KSUBSTRB("青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶", 18,24);  
    PUT "The substring found is: " result;  
RUN;
```

This produces the following output:

```
The substring found is: 花色 、 猩々緋 青
```

Note:

If you had specified 19 as that start byte, no string would have been returned as that start position would have split the byte representing a character.

Example – returning an error

In this example, the function is used to find and return the string that begins at byte 38 and is 28 bytes long. The result is written to the log.

```
DATA _NULL_;  
    result=KSUBSTRB("青碧 熨斗目花色 、 猩々緋 青柳鼠 ときがら茶", 38,28);  
    PUT "The substring found is: " result;  
RUN;
```

This produces the following output:

```
NOTE: Argument 3 to function KSUBSTRB at line 3669 column 11 is invalid  
The substring found is: 青柳鼠 ときがら茶
```

SCAN

Returns the word at a specified index position in a string. The separator between words can be specified, if required. Various modifiers can be also be used to specify additional delimiters and to modify the way the source string is searched.



The source string is treated as a list of words delimited by spaces (by default), or by any other character you specify. Each word occupies a position in the list from 1 through n , where n is the number of words in the specified string.

Return type: Character

string

Type: Character

The string to be examined.

n

Type: Numeric

The ordinal position of the word in the string you want to examine.

delimiters

Optional argument

Type: Character

A delimiter used to separate words. You can specify more than one delimiter.

modifiers

Optional argument

Parameters that specify characters to be used:

- As delimiters if the string contains none of the delimiters specified in *delimiter*
- In addition to the specified delimiters
- to modify the search in the string

You can specify more than one modifier. For example, you can specify 'C', or 'CAS'.

The following modifiers are available:

"A"

Alphabetic characters, or a string of characters, are considered delimiters.

"B"

Examines the string from right to left rather than left to right.

"C"

Control characters are considered delimiters.

"D"

Decimal numbers (0 through 9) are considered delimiters.

"F"

Characters considered valid characters for the first position in a variable name are considered delimiters.

"G"

Graphical characters are considered delimiters.

"H"

All spaces including tabs are considered delimiters. As with spaces, if the `H` modifier is used, punctuation characters and spaces are also considered delimiters.

"I"

Ignore the case of alphabetic characters specified as delimiters. For example, this modifier will make `A` equivalent to `a` as a delimiter.

"K"

Cause modifiers to act in the opposite way. For example, if `L` is set, instead of considering all lower-case characters as delimiters, the function will instead consider all uppercase characters as delimiters.

"L"

Lowercase alphabetic character will be considered a delimiter.

"M"

Multiple adjacent separators delimit nulls. By default, multiple adjacent separators are assumed to operate as a single separator. In this example, this option is not specified (the default):

```
DATA _NULL_;
  result1 = SCAN('london,,bike,company;;A1', 3, ',;');
  PUT result1;
RUN;
```

The function would return `company`, as the adjacent separators are treated as a single separator. However, if you specified the `M` modifier:

```
DATA _NULL_;
  result1 = QUOTE(SCAN('london,,bike,company;;A1', 3, ',;','M'));
  PUT result1;
RUN;
```

the function would return `" "`, as adjacent separators are now treated as individual separators each delimiting a null.

Note:

The `QUOTE` function has been used in this example so that the null can be seen in the result written to the log.

"N"

Characters considered valid for a variable name are considered delimiters.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"P"

Punctuation marks are considered delimiters. Punctuation marks are as defined in the collating sequence for the device.

"Q"

Ignore separators in strings delimited by quotation marks. For example:

```
data _null_;
  result1 = scan('The London Bike Company' A1 London      ',2,' ','q');
  result2 = scan('The London Bike Company' A1 London      ',2,' ');
  PUT result1;
  PUT result2;
RUN;
```

In this example, the following is written to the log:

```
A1
London
```

In the first use of the function, the first word is `The London Bike Company`, as the `Q` modifier ensures the spaces are not identified as separators; the second word is therefore `A1`. In the second use of the function, spaces within the quoted string are identified as separators, as the `Q` modifier is not specified; the second word is therefore `London`.

"R"

Remove opening and closing quotation marks from a string containing them.

"S"

Remove leading and trailing spaces from the returned string.

"T"

Remove trailing spaces from *delimiter* and from *source*.

"U"

Uppercase alphabetic character are considered delimiters.

"V"

Not currently available.

"W"

Printing characters are considered delimiters.

"X"

Characters that constitute a hexadecimal number (0-9, a-f, A-F) are considered delimiters.

Basic example

In this example, the function is used to find the fourth word in a string. The result is written to the log. No delimiter is specified, so spaces are used as delimiters.

```
DATA _NULL_;  
    result = SCAN('Steve Bike Company A 1 2 50 London',4);  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: A
```

Example – source with comma-delimited items

In this example, the function is used to find the fourth word in a string. The result is written to the log. No delimiter is specified, so the commas are recognised as delimiters.

```
DATA _NULL_;  
    result = SCAN('Steve,Bike,Company,A,1,2,50,London',4);  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: A
```

Example – source with various delimiters

In this example, the function is used to find the sixth word in a string. The result is written to the log. No delimiter is specified, so spaces and punctuation characters are recognised as delimiters; the `H` modifier has been used to ensure that horizontal tabs are also recognised as delimiters.

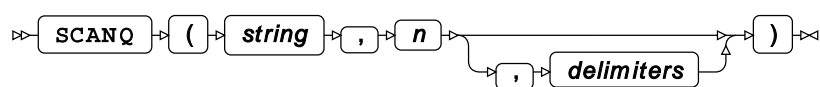
```
DATA _NULL_;  
    result = SCAN('Steve,Bike Company;A 1 2 50 London',8,, 'H');  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: London
```

SCANQ

Returns the word at a specified index position in a string. The separator between words can be specified if required.



The source string is treated as a list of words delimited by spaces (by default), or by any other character you specify. Each word occupies a position in the list from 1 through n , where n is the number of words in the specified string.

Note:

This function is similar to the `SCAN` function but has a simpler format; it is also, therefore, less flexible than `SCAN`. If you require more flexibility in searching the source string, see the `SCAN` function.

Return type: Character

string

Type: Character

The string to be examined.

n

Type: Numeric

The position of the word in the string you want to examine.

delimiters

Optional argument

Type: Character

A delimiter used to separate words. You can specify more than one delimiter.

If you do not set the *delimiter* argument, or set it to null (' '), spaces are recognised as delimiters. Tabs are not recognised as spaces.

Basic example

In this example, the function is used to find the fourth word in a string. The result is written to the log. No delimiter is specified, so spaces are used as delimiters.

```
DATA _NULL_;  
    result = SCANQ('Steve Bike Company A 1 2 50 London',4);  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: A
```

As no delimiter has been specified, spaces are used as delimiters; the function returns the string `A`.

Example – specifying delimiter

In this example, the function is used to find the fourth word in a string. The result is written to the log.

```
DATA _NULL_;  
    result = SCANQ('Steve,Bike,Company,A,1,2,50,London',4, ' ');  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: A
```

Example – specifying multiple delimiters

In this example, the function is used to find the sixth word in a string. The result is written to the log. Comma (,) and semi-colon (;) are specified as delimiters.

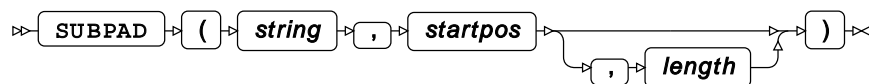
```
DATA _NULL_;  
    result = SCANQ('Steve,Bike Company;A 1 2 50 London',3,',';');  
    PUT "The word found is: " result;  
RUN;
```

This produces the following output:

```
The word found is: A 1 2 50 London
```

SUBPAD

Returns the substring of a specified length that starts at a specified position in a source string, and pads the returned substring to a specified length if required.



Finds a substring of a specified length in a source string that starts at a specified position. If the returned string is shorter than the length of substring you requested (if, for example, the end of the string is reached), padding is added in the form of trailing spaces.

Return type: Character

string

Type: Character

The source string.

startpos

Type: Numeric

The position in the source string at which to return the substring.

length

Optional argument

Type: Numeric

The number of characters of the source string to be returned.

If you do not specify *length*, the function returns the substring that begins at *position* and continues through to the end of the source string.

If *length* is set to 0 (zero), null (' ') is returned; unlike `SUBSTR`, no error is returned.

Example

In this example, the function is used to find and return the string that begins at position 35 and is ten characters long. The result is written to the log. The `QUOTE` function has been used to make the padding visible.

```
DATA _NULL_;  
  result=QUOTE(SUBPAD("Magnificent Bike Company A 1 2 50 London",35,10));  
  PUT 'The padded substring is: ' result;  
RUN;
```

This produces the following output:

```
The padded substring is: "London      "
```

Because the source string ends after the last character of the substring `London`, and that substring is only six characters long, the function returns `"London "`; four trailing spaces have been added to pad the result to the required ten character length.

SUBSTR

Returns either a substring of a specified length from a source string, or the string that results from replacement of a specified substring in a source string.



This function can be used in two ways:

- You can return a substring of a specified length from a source string. For example, you can return the substring that starts at the second character in the string `London`, and is three characters long: `ond`.
- You can return the string that results from replacing a specified substring in a source string with a specified string. For example, in the string `Linden` you could replace `inde` with `ondo` and return `London`.

Return type: Character

string

Type: Character

The source string.

startpos

Type: Numeric

The position in the source string at which to return the substring, or replace a substring

length

Optional argument

Type: Numeric

The number of characters of the source string to be returned, or to be replaced.

You return a substring by specifying a return variable on the left-hand side of the function. For example, `a = substr("London", 2, 2)`.

You replace a substring by specifying a replacement string on the right-hand side of the function. For example, `substr(x, 2, 6) = "ondo"`. If the function is used in this way, *string* must be a variable. The replacement is made in the specified variable. If the replacement string is shorter than the length specified, the replacement string is padded with spaces. If the replacement string is longer than the length specified, the replacement string is truncated appropriately. See the examples below.

If the combination of *startpos* and *length* is longer than the source string, an error occurs and a note is written to the log, but the program continues to run. Use [SUBPAD](#) (page 2000) or [SUBSTRN](#) (page 2004) if you want to return a substring, but do not want an error to occur. With *SUBPAD*, if the value of *length* causes the function to read beyond the end of the source string, the returned substring is padded with spaces to the specified length. With *SUBSTRN*, if *length* causes the function to read beyond the source string, the substring starting at the specified position and finishing at the end of the string is returned. For both functions, no error message is returned.

Example – returning substring

In this example, the function is used to find and return the string that begins at position 35 and is six characters long. The result is written to the log.

```
DATA _NULL_;  
    result = SUBSTR("Magnificent Bike Company A 1 2 50 London", 35,6);  
    PUT "The returned substring is: " result;  
RUN;
```

This produces the following output:

```
The returned substring is: "London"
```


Example – replacing substring

In this example, the function is used to replace the string that begins at position two and is four characters long. The result is written to the log.

```
DATA _NULL_;  
  name = "Linden";  
  SUBSTR(name, 2,4) = "ondo";  
  PUT "The new string is: " name;  
RUN;
```

This produces the following output:

```
The returned substring is: "London"
```

Example – replacing with a substring shorter than specified length

In this example, the function is used to replace the string that begins at position six and is four characters long. The result is written to the log.

```
DATA _NULL_;  
  name = "Linden";  
  SUBSTR(name, 2,4) = "YY";  
  PUT "The new string is: " name;  
RUN;
```

This produces the following output:

```
The new string is: LYY  n
```

Because the replacement string is only two characters long, but the length of the substring to be replaced is four characters, spaces have been used to pad the replacement string.

Example – replacing with a substring longer than specified length

In this example, the function is used to replace the string that begins at position six and is two characters long. The result is written to the log.

```
DATA _NULL_;  
  name = "London";  
  SUBSTR(name, 2,2) = "ABCDE";  
  PUT "The new string is: " name;  
RUN;
```

This produces the following output:

```
The new string is: LABdon
```

Because the replacement string is five characters long, but the length of string to be replaced is only two characters, the replacement string is truncated.

Example – returning an error

In this example, the function is used to find and return the string that begins at position 35 and is ten characters long. The result is written to the log.

```
DATA _NULL_;  
    result = SUBSTR("Magnificent Bike Company A 1 2 50 London", 35,10);  
    PUT "The returned substring is: " result;  
RUN;
```

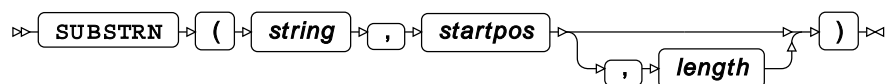
This produces the following output:

```
NOTE: Argument 3 to function SUBSTR at line 3296 column 11 is invalid  
The returned substring is: London  
_N_=1 _ERROR_=1 result=London
```

This example returns the string starting at the position specified through to the end of the string, and a message indicating that an error has occurred.

SUBSTRN

Returns a substring of a specified length from a specified starting point in a source string.



The substring can have zero length, and the function can also read beyond the end of the source string without error.

Return type: Character

string

Type: Character

The source string.

startpos

Type: Numeric

The position in the source string at which to return the substring of *length*.

length

Optional argument

Type: Numeric

The number of characters of the source string to be read.

If you do not specify *length*, the substring found starting at *position* through to the end of the source string is returned.

If *length* would result in an attempt to read over the end of the string, the characters from *position* through to the end of the string are returned. Unlike `SUBSTR`, no error is returned; unlike `SUBPAD`, no padding is appended.

If *length* is set to 0 (zero), null (' ') is returned; unlike `SUBSTR`, no error is returned.

Basic example

In this example, the function is used to find and return the string that begins at position 35 and is six characters long. The result is written to the log.

```
DATA _NULL_;
  result = SUBSTRN("Magnificent Bike Company A 1 2 50 London", 35,6);
  PUT "The returned substring is: " result;
RUN;
```

This produces the following output:

```
The returned substring is: London
```

Example – with zero length

In this example, the function is used to find and return the string that begins at position 35 and is zero characters long. The result is written to the log. The `QUOTE` function is also used in this example to help visualise the result.

```
DATA _NULL_;
  result=QUOTE(SUBSTRN("Magnificent Bike Company A 1 2 50 London", 35,0));
  PUT "The returned substring is: " result;
RUN;
```

This produces the following output:

```
The returned substring is: ""
```

Unlike `SUBSTR`, no error message is returned.

Example – reading over end of line

In this example, the function is used to find and return the substring that is ten characters long, beginning at position 35 in a string that is 40 characters long. The result is written to the log.

```
DATA _NULL_;
  result=SUBSTRN("Magnificent Bike Company A 1 2 50 London",35,10);
  PUT "The returned substring is: " result;
RUN;
```

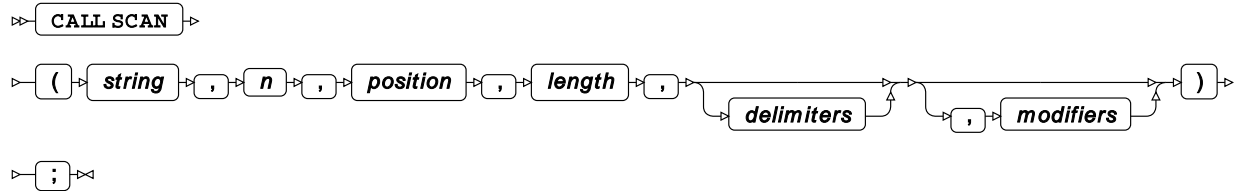
This produces the following output:

```
The returned substring is: London
```

Unlike `SUBSTR`, no error message is returned for exceeding the end of the source string. The function has read from the starting position to the end of the source string, and returned only that substring. No trailing spaces have been appended; you would need to use `SUBPAD` to achieve that result.

CALL SCAN

Returns the position and/or length of the substring at the specified index position in a source string.



The position and/or length are returned to variables specified in the routine. The source string is treated as a list of substrings delimited by spaces (by default) or any other character you specify. Each word occupies a position in the list from 1 through *n*. Various modifiers can be used to specify additional delimiters and to modify the way in which the source string is searched.

string

Type: Character

The string to be examined.

n

Type: Numeric

The position of the word you want to examine in the string.

position

Type: Numeric

The argument into which the character position of the substring in the source string is returned.

Note:

The delimiters are included in the character count for *position*.

length

Type: Numeric

The argument into which the length of the substring is returned. The length is the number of characters it comprises from its first character to the next delimiter.

delimiters

Optional argument

Type: Character

A delimiter used to separate words. You can specify more than one delimiter.

modifiers

Optional argument

Specifies characters to be used as delimiters if the string contains none of the delimiters specified in *delimiter*, or in addition to the specified delimiters, or to modify the search in the string. You can specify more than one modifier. For example, you can specify 'C', or 'CAS'.

The following modifiers are available:

"A"

Alphabetic characters, or a string of characters, are considered delimiters.

"B"

Examines the string from right to left rather than left to right.

"C"

Control characters are considered delimiters.

"D"

Decimal numbers (0 through 9) are considered delimiters.

"F"

Characters considered valid characters for the first position in a variable name are considered delimiters.

"G"

Graphical characters are considered delimiters.

"H"

All spaces including tabs are considered delimiters. As with spaces, if the H modifier is used, punctuation characters and spaces are also considered delimiters.

"I"

Ignore the case of alphabetic characters specified as delimiters. For example, this modifier will make A equivalent to a as a delimiter.

"K"

Cause modifiers to act in the opposite way. For example, if L is set, instead of considering all lower-case characters as delimiters, the function will instead consider all uppercase characters as delimiters.

"L"

Lowercase alphabetic character will be considered a delimiter.

"M"

Multiple adjacent separators delimit nulls. By default, multiple adjacent separators are assumed to operate as a single separator. In this example, this option is not specified (the default):

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('london,,,bike,company;;A1', 3, pos, len, ',,;');
  PUT pos= len=
RUN;
```

In this example, `pos` would be set to 15, and `len` to 7, as the adjacent separators (,,,) are treated as a single separator. Therefore, the third string found is `company`. However, if you specified the `M` modifier:

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('london,,,bike,company;;A1', 3, pos, len, ',,;', 'M');
  PUT pos= len=;
RUN;
```

`pos` would be set to 9, and `len` to 0, as adjacent separators (,,,) are now treated as individual separators each delimiting a null.

"N"

Characters considered valid for a variable name are considered delimiters.

"O"

Store the list of separators required the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"P"

Punctuation marks are considered delimiters. Punctuation marks are as defined in the collating sequence for the device.

"Q"

Ignore separators in strings delimited by quotation marks. For example:

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('"The London Bike Company" A1 London', 2, pos,
len,, 'q');
  PUT pos= len=;
  CALL SCAN('"The London Bike Company" A1 London', 2, pos, len,);
  PUT pos= len=;
RUN;
```

In this example, the following is written to the log:

```
pos=27 len=2  
pos=6 len=6
```

In the first use of the function, the first string is `The London Bike Company`, as the `Q` modifier ensures the spaces are not identified as separators; the second string is therefore `A1`. In the second use of the function, spaces within the quoted string are identified as separators, as the `Q` modifier is not specified; the second string is therefore `London`.

"R"

Remove opening and closing quotation marks from a string containing them.

"S"

Remove leading and trailing spaces from the returned string.

"T"

Remove trailing spaces from *delimiter* and from *source*.

"U"

Uppercase alphabetic character are considered delimiters.

"V"

Not currently available.

"W"

Printing characters are considered delimiters.

"X"

Characters that constitute a hexadecimal number (0-9, a-f, A-F) are considered delimiters.

If you do not specify a delimiter, spaces and punctuation characters are recognised as delimiters; in this case, spaces and punctuation characters can be mixed in the source string and will be recognised as delimiters. Tabs are not recognised as spaces, so if the source includes tab delimiters you need to use the `H` modifier; as with spaces, if the `H` modifier is used, punctuation characters and spaces will also be recognised as delimiters.

Basic example

In this example, the function is used to find the position and length of the third substring in a string. The result is written to the log. No delimiter is specified, so spaces are used as delimiters.

```
DATA _NULL_;  
pos=0;  
len=0;  
CALL SCAN('Magnificent Bike Company A 1 2 50 London',3,pos,len);  
PUT "The position of the third substring is " pos;  
PUT "The length of the third substring is " len;  
RUN;
```

This produces the following output:

```
The position of the third substring is 18
The length of the third substring is 7
```

The third substring in the source is Company

Example – source with comma-delimited items

In this example, the function is used to find the position and length of the eighth substring in a string. No delimiter is specified, so the commas in the source are recognised as delimiters. The result is written to the log.

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('Magnificent,Bike,Company,A,1,2,50,London',8,pos,len);
  PUT "The position of the eighth substring is " pos;
  PUT "The length of the eighth substring is " len;
RUN;
```

This produces the following output:

```
The position of the eighth substring is 35
The length of the eighth substring is 6
```

The eighth substring in the source is London.

Example – source with various delimiters

In this example, the function is used to find the sixth word in a string delimited by tabs. The result is written to the log. No delimiter is specified, so spaces and punctuation characters are recognised as delimiters; the H modifier has been used to ensure that horizontal tabs are also recognised as delimiters.

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('Magnificent,Bike,Company  A 1 2 50  London', 8, pos, len,, 'H');
  PUT "The position of the eighth substring is " pos;
  PUT "The length of the eighth substring is " len;
RUN;
```

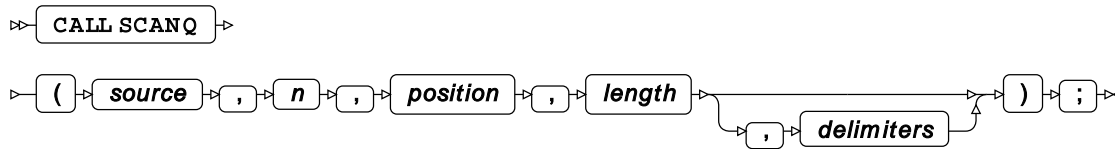
This produces the following output:

```
The position of the eighth substring is 39
The length of the eighth substring is 6
```

The eighth substring in the source is London (as commas, spaces and tabs have been recognised as delimiters).

CALL SCANQ

Returns the position and/or length of the substring at the specified index position in a source string, specifying the separator between substrings.



The position and/or length are returned to variables specified in the routine. The source string is treated as a list of words delimited by spaces (by default) or any other character you specify. Each word occupies a position in the list from 1 through *n*.

source

Type: Character

The string to be examined.

n

Type: Numeric

The position of the word you want to examine in the string.

position

Type: Numeric

The argument into which the character position of the substring in the source string is returned.

Note:

The delimiters are included in the character count for *position*.

length

Type: Numeric

The argument into which the length of the substring is returned. The length is the number of characters it comprises from its first character to the next delimiter.

delimiters

Optional argument

Type: Character

A delimiter used to separate words. This argument is optional. You can specify more than one delimiter.

If you do not set the *delimiter* argument, or set it to null (' '), spaces are recognised as delimiters. Tabs are not recognised as spaces. Multiple adjacent delimiters are recognised as one delimiter, but each adjacent delimiter is included in the character count.

Basic example

In this example, the function is used to find the position and length of the third substring in a string. This produces the following output: No delimiter is specified, so spaces are used as delimiters.

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCAN('Magnificent Bike Company A 1 2 50 London',3,pos,len);
  PUT "The position of the third substring is " pos;
  PUT "The length of the third substring is " len;
RUN;
```

This produces the following output:

```
The position of the third substring is 18
The length of the third substring is 7
```

The third substring in the source is Company

Example – specifying delimiter

In this example, the function is used to find the position and length of the third substring in the source string. The result is written to the log.

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCANQ('Magnificent,Bike,Company A 1 2 50 London',3,pos,len,',');
  PUT "The position of the third substring is " pos;
  PUT "The length of the third substring is " len;
RUN;
```

This produces the following output:

```
The position of the third substring is 18
The length of the third substring is 23
```

As the delimiter has been specified as the comma, spaces are not used as delimiters. The third substring in the source is, therefore, Company A 1 2 50 London.

Example – specifying multiple delimiters

In this example, the function is used to find the position and length of the fifth substring in a source string delimited using various punctuation marks. The result is written to the log. In this example, hyphen (-), full stop (.) and colon (:) are specified as delimiters.

```
DATA _NULL_;
  pos=0;
  len=0;
  CALL SCANQ('Magnificent-Bike-Company.A:1:2:50.London',5,pos,len,'-:');
  PUT "The position of the fifth substring is " pos;
  PUT "The length of the fifth substring is " len;
RUN;
```

This produces the following output:

```
The position of the fifth substring is 28
The length of the fifth substring is 1
```

In this example, the fifth substring is 1, and is one character long.

Find first character of a type

Return the first character of specified type in a source string.

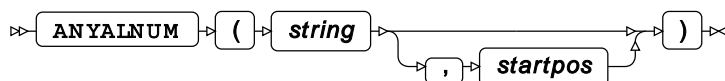
Character types can be alphanumeric character, punctuation, control and so on.

ANYALNUM ↗	2014
Returns the position of the first alphanumeric character in a string.	
ANYALPHA ↗	2015
Returns the position at which the first alphabetic character is found in a specified string.	
ANYCNTRL ↗	2016
Returns the position at which the first control character is found in a specified string.	
ANYDIGIT ↗	2018
Returns the position at which the first numeric character is found in a specified string.	
ANYFIRST ↗	2019
Returns the position at which the first character that matches the format of the first character of a variable name is found in a specified string.	
ANYGRAPH ↗	2020
Returns the position at which the first graphical character is found in a specified string.	
ANYLOWER ↗	2021
Returns the position at which the first lower case character is found in a specified string.	
ANYNAME ↗	2022
Returns the position of the first character in a string that could be a character in a variable name.	
ANYPRINT ↗	2023
Returns the position at which the first print character is found in a specified string.	
ANYPUNCT ↗	2024
Returns the position at which the first punctuation character is found in a specified string.	
ANYSpace ↗	2025
Returns the position at which the first space character is found in a specified string.	
ANYUPPER ↗	2026
Returns the position at which the first uppercase character is found in a specified string.	
ANYXDIGIT ↗	2028
Returns the position at which the first character that could constitute a hexadecimal number (0-9, a-f, A-F) is found in a specified string.	

FIRST ↗	2029
Returns the first character of a specified string.	
NOTALNUM ↗	2029
Returns the position in a string at which the first non-alphanumeric character is found.	
NOTALPHA ↗	2030
Returns the position of the first non-alphabetic character in a string.	
NOTCNTRL ↗	2031
Returns the position of the first non-control character in a string.	
NOTDIGIT ↗	2033
Returns the position of the first non-numeric character in a string.	
NOTFIRST ↗	2034
Returns the position of the first character in a string that cannot be used as the first character of a variable name.	
NOTGRAPH ↗	2035
Returns the position of the first character that is not a graphical character in a string.	
NOTLOWER ↗	2036
Returns the position of the first character in a string that is not a lower-case character .	
NOTNAME ↗	2037
Returns the position of the first character in a string that cannot be a character in a variable name.	
NOTPRINT ↗	2038
Returns the position in a string of the first character that is not a print character.	
NOTPUNCT ↗	2039
Returns the position in a string of the first character that is not a punctuation character.	
NOTSPACE ↗	2040
Returns the position in a string of the first character that is not a space character.	
NOTUPPER ↗	2042
Returns the position in a string of the first character that is not in uppercase.	
NOTXDIGIT ↗	2043
Find the position in a string of the first character that is not a hexadecimal number (that is, not one of the characters 0 through 9, a through f, and A through F). .	

ANYALNUM

Returns the position of the first alphanumeric character in a string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2..

Return type: Numeric

If no such character is found in the string, the value 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYALNUM("A101 01aSDx", 5)` returns 7, the position of the first alphanumeric after character position 5 (the space); `ANYALNUM("A101 01aSDx", -5)` returns 4, the position of the last alphanumeric before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first instance of an alphanumeric in a string, starting at the fourth character in the string. The result is written to the log.

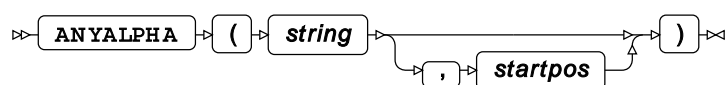
```
DATA _NULL_;  
    result = ANYALNUM("101 01adx", 4);  
    PUT "First alphanumeric found at position: " result;  
RUN;
```

This produces the following output:

```
First alphanumeric found at position: 6
```

ANYALPHA

Returns the position at which the first alphabetic character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2..

Return type: Numeric

If no such character is found in the string, the value 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYALPHA("A101 01aSDx", 5)` returns 9, the position of the first alphabetic character after position 5 (the space); `ANYALPHA("A101 01aSDx", -5)` returns 1, the position of the last alphabetic character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first alphabetic character, starting from the fifth character. The result is written to the log.

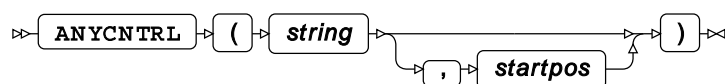
```
DATA _NULL_;  
  string1 = "0101  adxG" ;  
  result=ANYALPHA(string1, 5);  
  PUT "First alphabetic found at position: " result;  
RUN;
```

This produces the following output:

```
First alphabetic found at position: 8
```

ANYCNTRL

Returns the position at which the first control character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2..

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYCNTRL('33330A33330A'x , 5)` returns 6, the position of the first control character after character position 5; `ANYCNTRL('33330A33330A'x , -5)` returns 3, the position of the last control character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first control character. The result is written to the log.

```
DATA _NULL_;  
  a='33330A'x;  
  result=ANYCNTRL(a);  
  PUT "First control character found at position: " result;  
RUN;
```

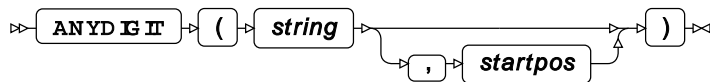
This produces the following output:

```
First control character found at position: 3
```

This is the position of the LF character.

ANYDIGIT

Returns the position at which the first numeric character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYDIGIT("A101 01aSDx", 5)` returns 7, the position of the first number after position 5 (the space); `ANYALPHA("A101 01aSDx", -5)` returns 4, the position of the last number before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first numeric character, starting from the fourth character. The result is written to the log.

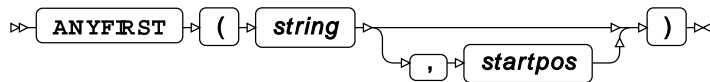
```
DATA _NULL_;
  string1="101 01adx";
  result = ANYDIGIT(string1, 4);
  PUT "First digit found at position: " result;
RUN;
```

This produces the following output:

```
First digit found at position: 6
```


ANYFIRST

Returns the position at which the first character that matches the format of the first character of a variable name is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYFIRST("A101 01aSDx", 5)` returns 9, the position of the first character, after position 5 (the space), that can be used as the first character in a variable name; `ANYFIRST("A101 01aSDx", -5)` returns 1, the position of the last character, before character position 5, that can be used as the first character in a variable name.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first character that can start a variable name. The result is written to the log.

```
DATA _NULL_;
  string1 = "0101  adxG" ;
  result=ANYFIRST(string1);
  PUT "Character that can be first in variable name found at position: " result;
RUN;
```

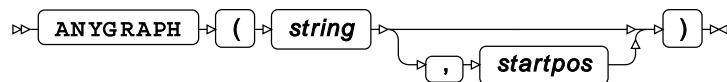
This produces the following output:

```
Character that can be first in variable name found at position: 8
```

Numbers and spaces cannot be the first character of a variable name.

ANYGRAPH

Returns the position at which the first graphical character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

>

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYGRAPH('1C1C311C1C31'x, 5)` returns 6, the position of the first graphical character after position 5; `ANYGRAPH('1C1C311C1C31'x, -5)` returns 3, the position of the last graphical character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first graphical character. The result is written to the log.

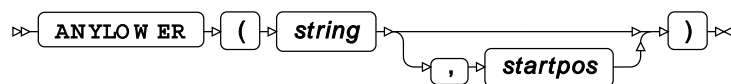
```
DATA _NULL_;  
  a='1C1C31'x;  
  result=ANYGRAPH(a);  
  PUT "First graphic character found at position: " result;  
RUN;
```

This produces the following output:

```
First graphic character found at position: 3
```

ANYLOWER

Returns the position at which the first lower case character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `anylower("A1b01 01aSDx", 5)` returns 10, the position of the first lowercase after position 5 (the space); `anylower("A1b01 01aSDx", -5)` returns 3, the position of the last lowercase character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first lowercase character, starting from the fifth character in the string. This produces the following output:

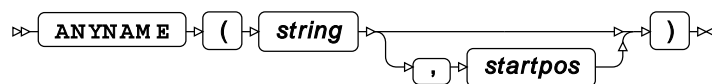
```
DATA _NULL_;  
  string1 = "0101  adxG" ;  
  result=ANYLOWER(string1, 5);  
  PUT "First lowercase character found at position: " result;  
RUN;
```

This produces the following output:

```
First lowercase character found at position: 8
```

ANYNAME

Returns the position of the first character in a string that could be a character in a variable name.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYNAME("A101 01aSDx", 5)` returns 7, the position of the first character, after position 5 (the space), that can be used in any position except the first in a variable name; `ANYNAME("A101 01aSDx", -5)` returns 4, the position of the last character before character position 5 that can be used in any position except the first in a variable name.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first character that could be used in a variable name, starting from the fifth character in the string. The result is written to the log.

```
DATA _NULL_;  
    string1 = "0101   adxG" ;  
    result=ANYNAME(string1, 5);  
    PUT "Character that can exist in variable name found at position: " result;  
RUN;
```

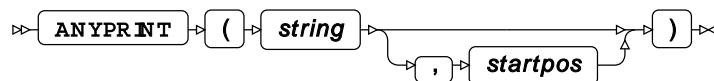
This produces the following output:

```
Character that can exist in variable name found at position: 8
```

The function will start searching the string from the fifth character in the string. The result will be 8, as spaces cannot be part of a variable name. If you had not specified that the search should start at the fifth character, but at the first, the result would have been 1 as numeric characters can be used in variable names (although not for the first character of a variable).

ANYPRINT

Returns the position at which the first print character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYPRINT('C1C01C1C311C1C31C1C010'x, 6)` returns 8, the position of the first printing character after position 6; `ANYPRINT('C1C01C1C311C1C31C1C010'x, -6)` returns 5, the position of the last printing character before the sixth character position.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first print character. The result is written to the log.

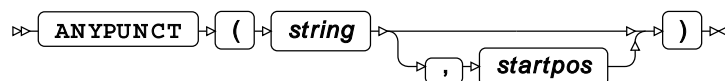
```
DATA _NULL_;  
  a='C1C010'x;  
  result=ANYPRINT(a);  
  PUT "First printing character found at position: " result;  
RUN;
```

This produces the following output:

```
First printing character found at position: 1
```

ANYPUNCT

Returns the position at which the first punctuation character is found in a specified string.



Punctuation characters are defined by the translation table in use.

You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYPUNCT("A1,01 01aS;Dx", 5)` returns 12, the position of the first punctuation character after position 5 (the space); `ANYPUNCT("A1,01 01aS;Dx", -5)` returns 3, the position of the last punctuation character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first punctuation character, starting from the fifth character in the string. The result is written to the log.

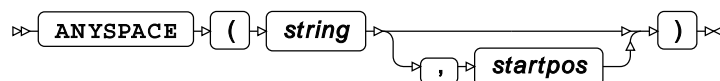
```
DATA _NULL_;  
  string1 = "0101ad;xG" ;  
  result=ANYPUNCT(string1, 5);  
  PUT "First punctuation character found at position: " result;  
RUN;
```

This produces the following output:

```
First punctuation character found at position: 7
```

ANYSPACE

Returns the position at which the first space character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYSPACE("A10 101a SDx", 5)` returns 9, the position of the first space after position 5; `ANYSPACE("A10 101a SDx", -5)` returns 4, the position of the last space before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Spaces include:

- Space character
- Horizontal tab
- Vertical tab
- Line feed
- Form feed

Example

In this example, the function searches the string for the first space character, starting from the third character in the string. The result is written to the log.

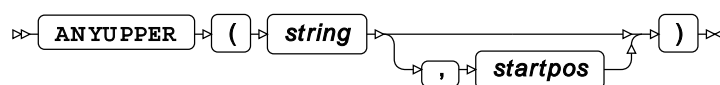
```
DATA _NULL_;  
  string1 = "0101   adxG" ;  
  result = ANYSPACE(string1, 3);  
  PUT "First space found at position: " result;  
RUN;
```

This produces the following output:

```
First space found at position: 5
```

ANYUPPER

Returns the position at which the first uppercase character is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYUPPER("A10 101a SDx", 5)` returns 10, the position of the first uppercase character after position 5 (the space); `ANYUPPER("A10 101a SDx", -5)` returns 1, the position of the last uppercase character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for an uppercase character starting at the fifth character. The result is written to the log.

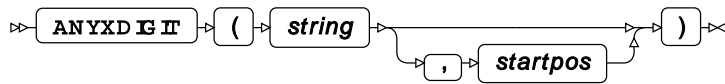
```
DATA _NULL_;
  string1 = "0101  adxG" ;
  result = ANYUPPER(string1, 5);
  PUT "First uppercase found at position: " result;
RUN;
```

This produces the following output:

```
First uppercase found at position: 11
```

ANYXDIGIT

Returns the position at which the first character that could constitute a hexadecimal number (0-9, a-f, A-F) is found in a specified string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no such character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `ANYXDIGIT("AB9 101a SDx", 4)` returns 7, the position of the first character that can be used in a hexadecimal number, after position 4; `ANYXDIGIT("AB9 101a SDx", -4)` returns 3, the position of the last character that can be used in a hexadecimal number, before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for a character that could be used in a hexadecimal number, starting at the fifth character. The result is written to the log.

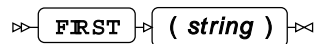
```
DATA _NULL_;
  string1 = "City  xA0" ;
  result=ANYXDIGIT(string1, 5);
  PUT "First hexadecimal character is found at position: " result;
RUN;
```

This produces the following output:

```
First hexadecimal character is found at position: 9
```

FIRST

Returns the first character of a specified string.



Return type: Character

string

Type: Character

The string or a variable containing the string to be examined.

If the string you specify is null (' '), a space character is returned.

Example

In this example, the function is used to find the first character in the source string. The result is written to the log.

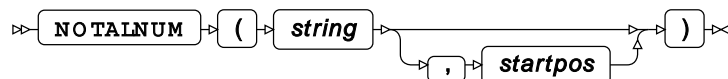
```
DATA _NULL_;  
    result = FIRST('Magnificent Bike Company Bike A 1 2 50    London');  
    PUT "The first character in the string is: " result;  
RUN;
```

This produces the following output:

```
The first character in the string is: M
```

NOTALNUM

Returns the position in a string at which the first non-alphanumeric character is found.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only alphanumeric characters are found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTALNUM("A101 01aSDx", 5)` returns 9, the position of the first character after position 5 that is not an alphanumeric; `NOTALNUM("A101 01aSDx", -5)` returns 4, the position of the last character before character position 5 that is not an alphanumeric.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches for the first instance of an alphanumeric in a string, starting at the tenth character in the string. The result is written to the log.

```
DATA _NULL_;  
    result = NOTALNUM("Magnificent Bike Company A 1 2 50 London", 10);  
    PUT "First non-alphanumeric found at position: " result;  
RUN;
```

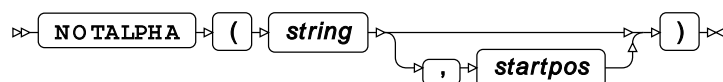
This produces the following output:

```
First non-alphanumeric found at position: 12
```

If the string had been "MagnificentBikeCompany", the result returned would be 0, as all characters in the string are alphanumeric.

NOTALPHA

Returns the position of the first non-alphabetic character in a string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no non-alphabetic character is found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTALPHA("AB9 AA1a SDx", 5)` returns 7, the position of the first character after position 5 that is not an alphabetic; `NOTALPHA("AB9 AA1a SDx", -5)` returns 4, the position of the last character that is not an alphabetic before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first non-alphabetic character. The result is written to the log.

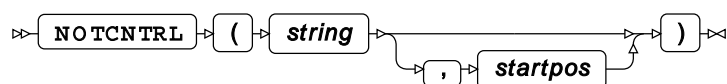
```
DATA _NULL_;
  string1 = "London0101 adxG" ;
  result = NOTALPHA(string1);
  PUT "First non-alphabetic found at position: " result;
RUN;
```

This produces the following output:

```
First non-alphabetic found at position: 7
```

NOTCNTRL

Returns the position of the first non-control character in a string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only control characters are found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTCNTRL('C1C01C1C311C1C31C1C010'x, 6)` returns 8, the position of the first character after position 5 that is not a control character; `NOTCNTRL('C1C01C1C311C1C31C1C010'x, -6)` returns 5, the position of the last character that is not an a control control character before character position 5 (that is, that position contains the last such character).

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first character that is not a control character. The result is written to the log.

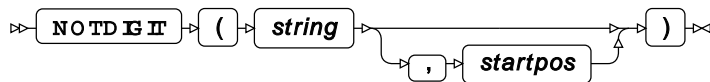
```
DATA _NULL_;
  a='0A3333'x;
  result = NOTCNTRL(a);
  PUT "First non-control character found at position: " result;
RUN;
```

This produces the following output:

```
First non-control character found at position: 2
```

NOTDIGIT

Returns the position of the first non-numeric character in a string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only numeric characters are found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTDIGIT("AB9 111a SDx", 6)` returns 8, the position of the first character after position 5 that is not a number; `NOTDIGIT("AB9 AA1a SDx", -5)` returns 4, the position of the last character that is not a number before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first non-numeric character. The result is written to the log.

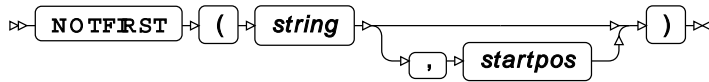
```
DATA _NULL_;  
    result = NOTDIGIT("101London 01adx");  
    PUT "First non-numeric found at position: " result;  
RUN;
```

This produces the following output:

```
First non-numeric found at position: 4
```

NOTFIRST

Returns the position of the first character in a string that cannot be used as the first character of a variable name.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If no matching characters are found, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTFIRST("A101AA AA1aSDx", 5)` returns 7, the position of the first character, after position 5 (the space), that cannot be used as the first character in a variable name; `NOTFIRST("A101AA AA1aSDx", -5)` returns 1, the position of the last character, before character position 4, that cannot be used as the first character in a variable name.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the position of the first character that can start a variable name, starting from the fifth character. The result is written to the log.

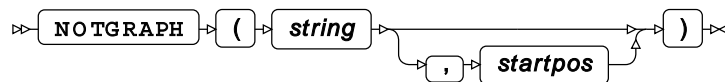
```
DATA _NULL_;
  string1 = "ad0101    xG" ;
  result = NOTFIRST(string1);
  PUT "First character that cannot be used as first character"
      " of variable name found at position: " result;
RUN;
```


This produces the following output:

```
First character that cannot be used as first character of variable name found at  
position: 3
```

NOTGRAPH

Returns the position of the first character that is not a graphical character in a string.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only graphical characters are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTGRAPH('1C1C45311C1C4531'x, 4)` returns 5, the position of the first non-graphical character after position 5; `NOTGRAPH('1C1C45311C1C4531'x, -4)` returns 2, the position of the last non-graphical character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

If only graphical characters are found in the string, the value 0 (zero) is returned.

Example

In this example, the function searches the string for the first non-graphical character, starting from the fifth character in the string. The result is written to the log.

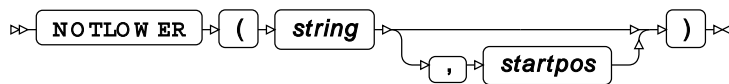
```
DATA _NULL_;  
  a='1C1C31'x;  
  result=notgraph(a);  
  PUT "First non-graphical character found at position: " result;  
RUN;
```

This produces the following output:

```
First non-graphical character found at position: 1
```

NOTLOWER

Returns the position of the first character in a string that is not a lower-case character .



You can specify a position at which to start searching the string, or start at the first character of the string (the default). For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only lower-case characters are found in the string, 0 (zero) is returned.

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, NOTLOWER("aa10a bA1aSDx", -7) returns 8, the position of the first non-lowercase character after position 7; NOTLOWER("aa10abA1aSDx", -7) returns 6, the position of the last non-lowercase character (in this case, the space) before character position 7.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for the first character that is not lowercase, starting from the fifth character in the string. The result is written to the log.

```
DATA _NULL_;  
  string1 = "bicycle, A1, 120";  
  result = NOTLOWER(string1,5);  
  PUT "First non-lowercase character found at position: " result;  
RUN;
```

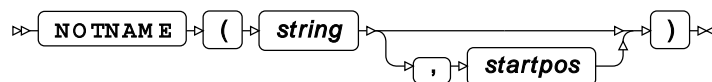
This produces the following output:

```
First non-lowercase character found at position: 8
```

This is the comma (,) after the word *bicycle*.

NOTNAME

Returns the position of the first character in a string that cannot be a character in a variable name.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only characters that can be used in a variable name are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTNAME("A 01SDx", 3)` returns 0, as there is no character after position 3, that cannot be used in any position except the first in a variable name; `NOTNAME("A 01SDx", -3)` returns 3, the position of the last character before character position 3 that cannot be used in any position except the first in a variable name.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for a character that could be used in a variable name, starting from the third character in the string. The result is written to the log.

```
DATA _NULL_;
  string1 = "ad0101  xG";
  result = NOTNAME(string1,3);
  PUT "First character that cannot be a character "
      "in a variable name found at position: " result;
RUN;
```

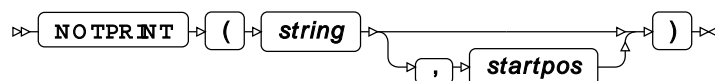
This produces the following output:

```
First character that cannot be a character in a variable name found at position: 7
```

This result is returned because spaces cannot be part of a variable name. If the starting position of the function had not been the third character, but the first; the result would still have been 7 as alphabetic and numeric characters *can* be used in variable names (although numerics *cannot* be used for the first character of a variable name).

NOTPRINT

Returns the position in a string of the first character that is not a print character.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only print characters are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTPRINT('1C1C45311C1C4531'x, 4)` returns 5, the position of the first non-printing character after position 4; `NOTPRINT('1C1C45311C1C4531'x, -4)` returns 2, the position of the last non-printing character before character position 4.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

If no print character is found in the string, the value 0 (zero) is returned.

Example

In this example, the function searches the string for the first non-printing character, starting from the fifth character in the string. The result is written to the log.

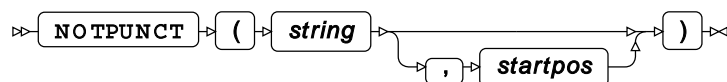
```
DATA _NULL_;  
  a='C1C010'x;  
  result=notprint(a);  
  PUT "First non-printing character found at position: " result;  
RUN;
```

This produces the following output:

```
First non-printing character found at position: 3
```

NOTPUNCT

Returns the position in a string of the first character that is not a punctuation character.



Punctuation characters are defined by the translation table in use.

You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only punctuation characters are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTPUNCT("A1,01...01aS", 7)` returns 9, the position of the first non-punctuation character after position 7; `NOTPUNCT("A1,01...01aS", -7)` returns 5, the position of the last non-punctuation character before character position 7.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

If only punctuation characters are found in the string, the value 0 (zero) is returned.

Example

In this example, the function searches the string for the first non-punctuation character, starting from the fifth character in the string. The result is written to the log.

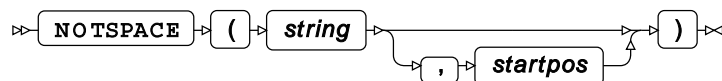
```
DATA _NULL_;  
  string1 = "#1,0,1ad;xG";  
  result = NOTPUNCT(string1, 5);  
  PUT "First non-punctuation character found at position: " result;  
RUN;
```

This produces the following output:

```
First non-punctuation character found at position: 6
```

NOTSPACE

Returns the position in a string of the first character that is not a space character.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only space characters are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTSPACE("A10 101a SDx", 9)` returns 11, the position of the first non-space character after position 9; `NOTSPACE("A10 101a SDx", -9)` returns 8, the position of the last non-space character space before character position 9.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

If only space characters are found in the string, the value 0 (zero) is returned.

Spaces include:

- Space character
- Horizontal tab
- Vertical tab
- Line feed
- Form feed

Example

In this example, the function searches the string for the first non-space character, starting from the ninth character in the string. The result is written to the log.

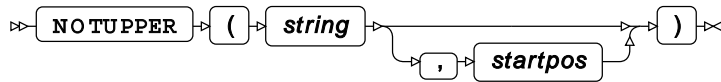
```
DATA _NULL_;  
  string1 = "10 01 01   adxG" ;  
  result= NOTSPACE(string1, 9);  
  PUT "First non-space found at position: " result;  
RUN;
```

This produces the following output:

```
First non-space found at position: 12
```

NOTUPPER

Returns the position in a string of the first character that is not in uppercase.



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only uppercase characters are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTUPPER("a10 ABCEG SDx", 5)` returns 10, the position of the first non-uppercase character after position 5; `NOTUPPER("a10 ABCEG SDx", -5)` returns 4, the position of the last non-uppercase character before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for a character not in uppercase. The result is written to the log.

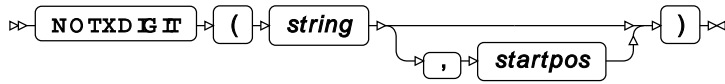
```
DATA _NULL_;
  string1 = "LONDON0101   adx G" ;
  result = NOTUPPER(string1);
  PUT "First non-uppercase character found at position: " result;
RUN;
```

This produces the following output:

```
First non-uppercase character found at position: 7
```


NOTXDIGIT

Find the position in a string of the first character that is not a hexadecimal number (that is, not one of the characters 0 through 9, a through f, and A through F). .



You can optionally specify a position at which to start searching the string. For example, if you want to start the search at the second position of the string, you set *startpos* to 2.

Return type: Numeric

If only characters that constitute hexadecimal numbers are found in the string, 0 (zero) is returned

string

Type: Character

The string to be analysed.

startpos

Optional argument

Type: Numeric

The position in *string* at which the search starts.

If you set *startpos* to a positive number, the function searches for the first instance of the character at or beyond that character position. If you set *startpos* to a negative number, the function looks backward through the string starting at the (positive) character position.

For example, `NOTXDIGIT("AB9H 101aH SDx", 6)` returns 10, the position of the first character that cannot be used in a hexadecimal number, after position 6; `NOTXDIGIT("AB9H 101aH SDx", -6)` returns 5, the position of the last character that cannot be used in a hexadecimal number, before character position 5.

If you do not specify a value for this argument, the search starts at the first (leftmost) character of *string*.

Example

In this example, the function searches the string for characters that do not make up hexadecimal numbers. The result is written to the log.

```
DATA _NULL_;
  string1 = "01A0London00Bicycle" ;
  result = NOTXDIGIT(string1);
  PUT "First non-hexadecimal character found at position: " result;
RUN;
```

This produces the following output:

```
First non-hexadecimal character found at position: 5
```

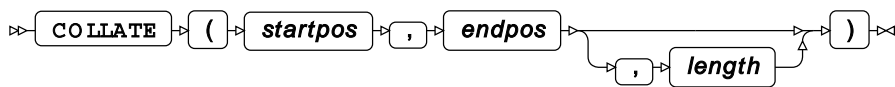
Find characters or rank in collating sequence

Return characters from specified positions in a collating sequence, or return the rank of a character in a sequence.

COLLATE ↗	2044
Returns the collating sequence between specified positions.	
RANK ↗	2045
Returns the rank of a character in the collating sequence of the device on which the function is executed.	

COLLATE

Returns the collating sequence between specified positions.



The string returned consists of the ASCII or EBCDIC collating sequence (depending on the device on which the function is run) from *startpos* to *endpos* inclusive.

Return type: Character

startpos

Type: Numeric

The position in the collating sequence at which the function should start returning characters, specified as an integer.

endpos

Type: Numeric

>

The position in the collating sequence at which the function should stop returning characters, specified as an integer. If you specify this argument, you do not need to specify *length*. If you omit this argument because you are specifying *length*, you must delimit the position with a comma (see examples below).

length

Optional argument

Type: Numeric

The number of characters to be returned. If you specify this argument, you do not need to specify *end-position*.

Note:

If you specify both *length* and *endpos*, the value of *endpos* will take precedence. For example, `result=COLLATE(65,90,10)` will return ABCDEFGHIJKLMNOPQRSTUVWXYZ rather than ABCDEFGHIJ.

Example – using specified start and end positions

In this example, the function is used to return the collating from position 65 through to position 122. The result is written to the log.

```
DATA _NULL_;  
    result=COLLATE(65,122);  
    PUT "The collating sequence is: " result;  
RUN;
```

This produces the following output:

```
The collating sequence is:  
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
```

This is the collating sequence for the specified starting position (65) through to the end position (122).

Example – specifying start position and number of characters

```
DATA _NULL_;  
    result=collate(65,,26);  
    PUT "The collating sequence is: " result;  
RUN;
```

This produces the following output:

```
The collating sequence is:  
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

This is the ASCII collating sequence from and including position 65 through the next 26 characters.

RANK

Returns the rank of a character in the collating sequence of the device on which the function is executed.

⇒ **RANK** ⇒ (*character*) ⇒

Return type: Numeric

character

Type: Character

The character for which you want the rank returned. (You can also specify a string, but the function will only examine the first character.)

Example

In this example, the rank of the character **A** in the current collating sequence is returned. The result is written to the log.

```
DATA _NULL_;
  char='A';
  result=RANK(char);
  PUT "Rank of character in the collating sequence: " result;
RUN;
```

This produces the following output:

```
Rank of character in the collating sequence: 65
```

Find position and length of substrings

Return the length and/or position of specified substrings in source strings.

CHAR ↗	2047
Returns the character found at the specified position in a string.	
CONTAINS ↗	2048
Returns a value identifying whether a specified string exists in another string.	
FIND ↗	2049
Returns the position of the first occurrence of a specified substring within a string.	
FINDC ↗	2051
Returns the position at which the specified character is first found within a string.	
INDEX ↗	2053
Returns the starting position of a substring within a specified string.	
INDEXC ↗	2054
Returns the first position at which a specified character occurs in a string.	
INDEXW ↗	2055
Returns the first position at which a specified word occurs in a string.	
KINDEX ↗	2057
Returns the starting position of a string within another string that consists of DBCS characters.	

KINDEXC ↗	2057
Returns the first position at which specified characters occur in a string that consists of DBCS characters.	
KLENGTH ↗	2058
Returns the length of a non-blank character string consisting of DBCS characters.	
KVERIFY ↗	2059
Returns a value indicating whether a character or string exists in another DBCS string.	
LENGTH ↗	2061
Returns the length of a character string, including leading spaces but excluding trailing spaces. If the string is null (' '), or contains only spaces, 0 (zero) is returned.	
LENGTHC ↗	2063
Returns the storage length of a character string. If the string is null, 1 is returned.	
LENGTHM ↗	2065
Returns the length of a string, including any leading or trailing spaces. If the string is null 1 is returned.	
LENGTHN ↗	2067
Returns the length of a character string, including leading spaces but excluding trailing spaces, and returns 0 (zero) if the string is null (' ') or contains only spaces.	
VERIFY ↗	2069
Returns the position of the first character of the source string that does not match a specified substring.	
WHICHC ↗	2071
Returns the position of the first string in a list of strings that matches a specified string.	

CHAR

Returns the character found at the specified position in a string.

➤ **CHAR** ➤ (*string* , *position*) ➤

Return type: Character

string

Type: Character

The string in which you want to find a character.

position

Type: Numeric

The position in the string at which to find the character.

Example

In this example, the function examines the string at the specified position, and returns the character at that position. The result is written to the log.

```
DATA _NULL_;  
    result = CHAR("101 01adx", 8);  
    PUT "The character at the specified position is: " result;  
RUN;
```

This produces the following output:

```
The character at the specified position is: a
```

CONTAINS

Returns a value identifying whether a specified string exists in another string.

➤ **CONTAINS** ➤ (*string* , *find-string*) ➤

Searches a source string for a specified string, and returns 1 if the string is found in the source string, 0 otherwise.

Return type: Numeric

string

Type: Character

The string to be examined.

find-string

Type: Character

The string to find in *source*.

Example – matching specified string

In this example, the function is used to find the string `London` in the source string. The result is written to the log.

```
DATA _NULL_;  
    search = 'London';  
    result = CONTAINS('Steve Bike Company A 1 2 50 London', search);  
    PUT "The result is: " result;  
RUN;
```

This produces the following output:

```
The result is: 1
```

The string `London` is found in the source string, therefore 1 is returned.

Example – not matching specified string

In this example, the function is used to find the string `london` in the source string. The result is written to the log.

```
DATA _NULL_;  
  search='london';  
  result=contains('Steve Bike Company A 1 2 50 London', search);  
  PUT "The result is: " result;  
RUN;
```

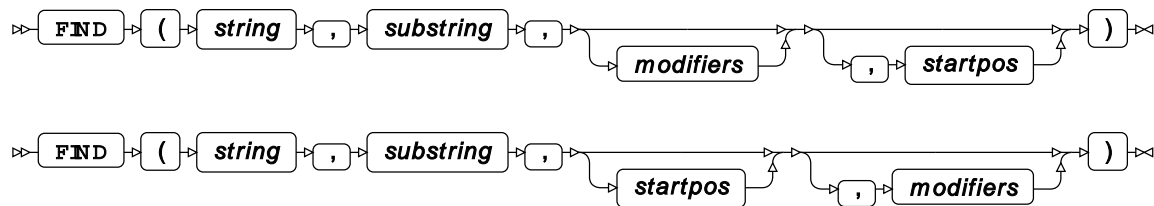
This produces the following output:

```
The result is: 0
```

The string `london` is not found in the source string, therefore 0 (zero) is returned.

FIND

Returns the position of the first occurrence of a specified substring within a string.



Return type: Numeric

If the source string does not contain an instance of the specified search string, the function returns 0.

string

Type: Character

The string in which you want to search for another string.

substring

Type: Character

The string for which you want to search in *string*.

modifiers

Optional argument

Parameters that modify the output. The following modifiers are available:

"I"

Ignore case.

"T"Trim spaces from the end of *substring*.**startpos**

Optional argument

Type: NumericThe position in *string* at which the function should start searching for *substring*.**Example – finding location of a string**

In this example, the function is used to find the first occurrence of the string *Bike* in the source string. The result is written to the log.

```
DATA _NULL_;
  search = 'Bike';
  result = FIND('Magnificent Bike Company A 1 2 50 (Bike) London', search);
  PUT "The string is found at character position: " result;
RUN;
```

This produces the following output:

```
The string is found at character position: 13
```

This is the position at which the first instance of the string *Bike* starts.

Example – ignoring case in search

In this example, the function is used to find the first occurrence of the string *Bike* or *bike* (or *bIke* or *BiKe*) in the source string. The result is written to the log.

```
DATA _NULL_;
  search = 'bike';
  result = FIND('Magnificent Bike Company A 1 2 50 London', search, 'I');
  PUT "The string is found at character position: " result;
RUN;
```

This produces the following output:

```
The string is found at character position: 13
```

In this case, *bike* matches *Bike* because the **I** modifier tells the function to ignore case.

Example – starting search at specified location

In this example, the function is used to find the first occurrence of the string 'Bike' in the source string from and including the character at the 20th position in the string. The result is written to the log.

```
DATA _NULL_;  
  search = 'Bike';  
  result = FIND('Magnificent Bike Company A 1 2 50   London', search, 20);  
  PUT "The string is found at character position: " result;  
RUN;
```

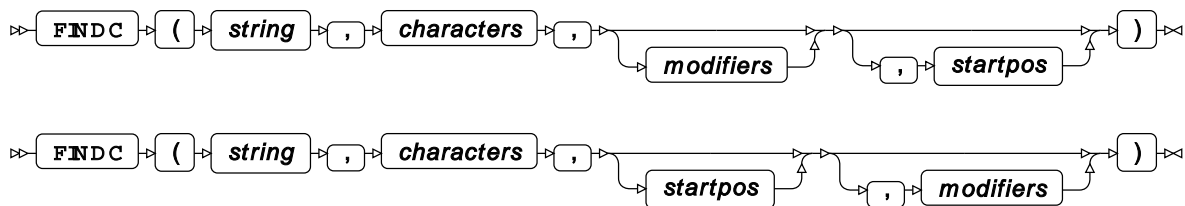
This produces the following output:

```
The string is found at character position: 0
```

The function starts searching the source string at the specified position, 20. The function doesn't find a match for the search string, because the search starts after the position of the only occurrence of the search string in the source string. The function therefore returns the value 0.

FINDC

Returns the position at which the specified character is first found within a string.



You can specify more than one character for the search, and the function will find the first occurrence of any character in the string.

Return type: Numeric

If the source string does not contain an instance of a specified character, 0 (zero) is returned.

string

Type: Character

The string in which you want to search for character.

characters

Type: Character

One or more characters for which you want to search in *string*.

modifiers

Optional argument

"I"

Ignore case.

"O"

Store the list of characters to be searched for the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"T"

Strips any trailing spaces in *characters*.

"V"

Return the value 1 to indicate that at least one of the search characters exists in the string.

startpos

Optional argument

Type: Numeric

The position in *string* at which the function should start searching for *characters*.

Example – simple find

In this example, the function is used to find any of the characters F5ox in the source string. The result is written to the log.

```
DATA _NULL_;  
  search = 'F5ox';  
  result = FINDC('Magnificent, Bike, Company A 1 2 50 London', search);  
  PUT "The character is first found at character position: " result;  
RUN;
```

This produces the following output:

```
The character is first found at character position: 21
```

This is the position at which the first instance of a character in the search string (in this example o) occurs.

Example – ignoring case in search

In this example, the function is used to find any of the characters F5ox in the source string, and ignores the case of the characters in the search string. The result is written to the log.

```
DATA _NULL_;  
  search = 'F5ox';  
  result = FINDC('Magnificent, Bike, Company A 1 2 50 London', search, 'I');  
  PUT "The character is first found at character position: " result;  
RUN;
```

This produces the following output:

```
The character is first found at character position: 6
```

This is the position at which the first instance of the `f` character occurs; this matches the search character `F` because the `I` modifier has been set.

Example – starting search at specified location

In this example, the function is used to find any the character `B` in the source string, starting from character position 20. The result is written to the log.

```
DATA _NULL_;  
  search = 'B';  
  result = FINDC('Magnificent Bike Company A 1 2 50   London', search, 20);  
  PUT "The character is first found at character position: " result;  
RUN;
```

This produces the following output:

```
The character is first found at character position: 0
```

Because the function starts searching the source string at the specified position, 20, the function finds no match for the search character.

Example – confirming existence of character

In this example, the function is used to find whether the character `B` exists in source string. The result is written to the log.

```
DATA _NULL_;  
  search = 'B';  
  result = FINDC('Magnificent Bike Company A 1 2 50   London', search, 'V');  
  PUT "The character is first found at character position: " result;  
RUN;
```

This produces the following output:

```
The character is first found at character position: 1
```

Because the `V` modifier has been specified, the function returns 1 to confirm that the character exists in the string, not the position at which the character was first was found.

INDEX

Returns the starting position of a substring within a specified string.

➤ **INDEX** ➤ (*string* , *substring*) ➤

Finds a specified substring within a string, and returns the position of that substring.

Return type: Numeric

If the substring is not found, 0 (zero) is returned.

string

Type: Character

A string to be examined for a substring.

substring

Type: Character

The substring to be found in *source*.

Example

In this example, the function is used to find a substring in a source string. The result is written to the log.

```
DATA _NULL_;
  search='Bike';
  result = INDEX('Magnificent Bike Company Bike A 1 2 50   London', search);
  PUT "The string is found at character position: " result;
RUN;
```

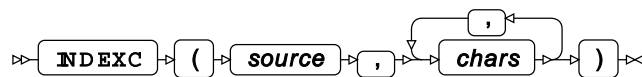
This produces the following output:

```
The string is found at character position: 13
```

This is the first position at which the string was found.

INDEXC

Returns the first position at which a specified character occurs in a string.



You can specify more than one character, in which case the position of the first matching character is found. For example, if you specify a search for the character *i* in the string *Bike*, the function returns 2, which is the position of the first occurrence of that character. However, if you search for *iB*, the function returns 1, which is the position of the first of the characters found, in this case *B*.

Return type: Numeric

If the character is not found, 0 (zero) is returned.

source

Type: Character

A string be examined for one or more characters.

chars**Type:** CharacterOne or more characters to be searched for in *source*.**Example**

In this example, the function is used to find one of the characters *i* (*excerpt*) in a source string. The result is written to the log.

```
DATA _NULL_;
  search='bko';
  result = indexc('Magnificent Bike Company Bike A 1 2 50   London', search);
  PUT "The first character is found at position: " result;
RUN;
```

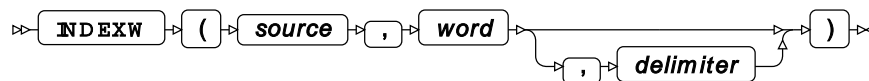
This produces the following output:

```
The first character is found at position: 15
```

This is the first position at which a matching character was found; in this example, *k*. Case must match, therefore *b* was not matched with the *B* in *Bike*.

INDEXW

Returns the first position at which a specified word occurs in a string.



By default, a word is a string delimited by spaces. You can, however, specify the delimiter that is used in the string. A word must be completely delimited to be found; for example, *Bike* will not be matched in the source *MagnificentBike*, as *MagnificentBike* is the entire delimited word; see examples below for more detail.

Return type: Numeric

If the word is not found, 0 (zero) is returned.

source**Type:** Character

A string to be examined for an occurrence of a word.

word**Type:** CharacterThe word to be searched for in *source*.

delimiter

Optional argument

Type: Character

The character used as the separator in *source*.

Basic example

In this example, the function is used to find the position at which the word `Bike` is first found in a source string. The result is written to the log.

```
DATA _NULL_;
  search='Bike';
  result = INDEXW('Magnificent Bike Company (Bike)', search);
  PUT "The word is found at character position: " result;
RUN;
```

This produces the following output:

```
The word is found at character position: 13
```

This is the position at which the first occurrence of the word `Bike` was found. Case must match; if `bike` had been specified in search, it would not have matched `Bike` and 0 would have been returned.

Example – using a separator

In this example, the function is used to find the position at which the word `Bike` in a source string; words in the source are separated by commas (.). The result is written to the log.

```
DATA _NULL_;
  search='Bike';
  separator=',';
  result = INDEXW('Magnificent Bike Company,Bike,London', search, separator);
  PUT "The word is found at character position: " result;
RUN;
```

This produces the following output:

```
The word is found at character position: 26
```

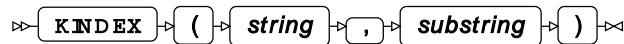
This is the position at which the first occurrence of the word `Bike` was found. In this example, the string `Magnificent Bike Company` is delimited by a comma, and is thus seen as one word. If the string had been:

```
Magnificent Bike Company, Bike,London
```

no match would have been found as a word is delimited by separators, and the word `Bike` in this case is preceded by a space which would be included as part of the word.

KINDEX

Returns the starting position of a string within another string that consists of DBCS characters.



Finds the starting point of a substring within a source string that consists of characters from a double-byte character set (DBCS). If the substring is not found, zero (0) is returned.

Return type: Numeric

string

Type: Character

A string to be examined for a substring.

substring

Type: Character

The substring to be found in *source*.

Example

In this example, the function is used to find the position at which the substring ときから茶 is found in a source string. The result is written to the log.

```
DATA _NULL_;
  search='ときから茶';
  result = KINDEX('青碧 熨斗目花色 ときから茶 猩々緋 青柳鼠', search);
  PUT result;
RUN;
```

This produces the following output:

```
10
```

which is the first position at which the string was found.

KINDEXC

Returns the first position at which specified characters occur in a string that consists of DBCS characters.



Finds a specified character or string (an *excerpt*) within a string consisting of characters from double-byte character set (DBCS), and returns the position at which the excerpt is first found. If the excerpt is not found, 0 (zero) is returned.

Return type: Numeric

string

Type: Character

A string to be examined for an excerpt.

chars

Type: Character

A character or string to be searched for in *source*.

Example

In this example, the function is used to find the position at which one of the characters と斗 is first found in a source string. The result is written to the log.

```
DATA _NULL_;  
  search='と斗';  
  result = KINDEXC('青碧 熨斗目花色 ときがら茶 猩々緋 青柳鼠', search);  
  PUT result;  
RUN;
```

This produces the following output:

```
5
```

This is the position at which 斗 is first found.

KLENGTH

Returns the length of a non-blank character string consisting of DBCS characters.

⇒ **KLENGTH** (*string*) ⇒

Returns the length of a non-blank character string consisting of characters from a double-byte character set (DBCS), excluding any trailing blanks. If the string is blank (null, or composed entirely of spaces), the function returns the value 1. If a number is specified as an argument, a value of 12 is returned, and a message written to the log noting that numeric values have been converted to character values.

Return type: Numeric

string**Type:** Character

The string to be examined.

Example

In this example, the function is used to find the length of various strings. The result is written to the log.

```
DATA _NULL_;  
  number=123456789012345678;  
  result1=KLENGTH('青碧 熨斗目花色 ときから茶 猩々緋 青柳鼠  ');  
  result2=KLENGTH('      ');  
  result3=KLENGTH(number);  
  
  PUT result1;  
  PUT result2;  
  PUT result3;  
run;
```

This produces the following output:

```
22 0 12
```

In this example:

- The first use of the function returns the value 22, which is the length of the string without trailing blanks.
- The second use of the function returns the value 0, as the string contains only blanks.
- The third use of the function returns the value 12, because the string contains a numeric value. A message is written to the log noting that numeric values have been converted to character values.

KVERIFY

Returns a value indicating whether a character or string exists in another DBCS string.



Searches a source string, consisting of characters from a double-byte character set, and verifies whether one or more substrings (*excerpts*) exist or not in that string. The function returns the position of the first character in the source string that does not match the excerpt. If the source string contains none of the specified excerpts 0 (zero) is returned.

Return type: Numeric**string****Type:** Character

The string to be searched for specified substrings.

excerpt

Type: Character

A character or substring to be found in *source*.

Example – checking that a string contains a character

In this example, the function searches for the character 青. The result is written to the log.

```
DATA _NULL_;  
    result = KVERIFY('青碧 熨斗目花色 猩々緋 青柳鼠 ときがら茶', '青');  
    PUT 'The character is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 2
```

In this example, the function searches for the character 碧. The result is written to the log.

```
DATA _NULL_;  
    result = KVERIFY('青青青青青青碧碧碧碧碧', '碧');  
    PUT 'The character is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 1
```

In this example, the function searches for the characters 碧 and 青. The result is written to the log.

```
DATA _NULL_;  
    result = KVERIFY('青青青青青青碧碧碧碧碧', '碧','青');  
    PUT 'The character is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 0
```

The function returns 0 (zero) in this instance as there is nowhere in the string where one of the characters does not exist.

Example – checking that a string contains a substring

In this example, the function searches for the string 熨斗目花色. The result is written to the log.

```
DATA _NULL_;  
    result = KVERIFY('青碧 熨斗目花色 猩々緋 青柳鼠 ときがら茶', '熨斗目花色');  
    PUT 'The string is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The string is first NOT found at position: 1
```

In this example, the function searches for the strings 青碧 and 熨斗目花色. The result is written to the log.

```
DATA NULL;  
  RESULT = KVERIFY('青碧 熨斗目花色 猩々緋 青柳鼠 ときから茶', '青碧', '熨斗目花色');  
  PUT 'The strings are first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The strings are first NOT found at position: 15
```

LENGTH

Returns the length of a character string, including leading spaces but excluding trailing spaces. If the string is null (' '), or contains only spaces, 0 (zero) is returned.

➤ **LENGTH** ➤ (*string*) ➤

If a number is specified as an argument, a value of 12 is returned, and a message written to the log noting that numeric values have been converted to character values.

Return type: Numeric

string

Type: Character

A string to be examined for a substring.

If the length of a variable has been specified and the variable contains a string to be checked, then if the string is longer than the specified length, the length of the variable will be returned.

Basic example

In this example, the function is used to find the length of various strings. The result is written to the log.

```
DATA _NULL_;  
  number = 1234567890123456789;  
  result1 = LENGTH('Magnificent Bike Company Bike A 1 2 50   London   ');  
  result2 = LENGTH('   ');  
  result3 = LENGTH('');  
  result4 = LENGTH(number);  
  
  PUT "The length of the string is: " result1;  
  PUT "The length of the string is: " result2;  
  PUT "The length of the string is: " result3;  
  PUT "The length of the string is: " result4;  
RUN;
```

This produces the following output:

```
The length of the string is: 51  
The length of the string is: 0  
The length of the string is: 0  
The length of the string is: 12
```

In this example:

- The first use of the function returns the value 51, which is the length of the string ignoring the trailing blanks.
- The second use of the function returns the value 0, as the string contains only spaces.
- The third use of the function returns the value 0, as the string contains a null.
- The fourth use of the function returns the value 12, because the string contains a numeric value. A message is written to the log by WPS noting that numeric values have been converted to character values.

Example – strings in variables of a specified length

In this example, the function is used to find the length of various strings that have been assigned to variables with a specified length. The result is written to the log.

```
DATA _NULL_;
  LENGTH twsl twsln $10;
  twsl = "Bicycle";
  result = LENGTH(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsl = "Twenty-three bicycles";
  result = LENGTH(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsln=" ";
  result = LENGTH(twsln);
  ps = QUOTE(twsln);
  PUT "The string is now: " ps " and its length is: " result;

RUN;
```

This produces the following output:

```
The string is now: "Bicycle  "   and its length is: 7
The string is now: "Twenty-thr" and its length is: 10
The string is now: "          " and its length is: 1
```

Because this function excludes trailing spaces, the padding spaces applied to the strings to make them match the length specified for the variable are ignored. However, the string that is longer than the specified variable length has been truncated, and the length returned by the function is the same as the variable length.

Contrast this result to that obtained with [LENGTHC](#) (page 2063), [LENGTHM](#) (page 2065) and [LENGTHN](#) (page 2067).

LENGTHC

Returns the storage length of a character string. If the string is null, 1 is returned.

➤ **LENGTHC** ➤ (*string*) ➤

If a number is specified as an argument, the value 12 is returned, and a message written to the log noting that numeric values have been converted to character values.

The storage length is set when the program is compiled. If you explicitly set the length of the variable using **LENGTH** before assigning a value to it, then the value returned will be the length specified by **LENGTH**. If you assign the value before specifying **LENGTH**, the value returned will be length of the string (excluding trailing spaces). See examples below.

Return type: Numeric

string

Type: Character

A string.

Basic example

In this example, the function is used to find the length of various strings. The result is written to the log.

```
DATA _NULL_;
  number = 1234567123456789;
  result1 = LENGTHC('Magnificent Bike Company Bike A 1 2 50   London   ');
  result2 = LENGTHC('   ');
  result3 = LENGTHC('');
  result4 = LENGTHC(number);

  PUT "The length of the string is: " result1;
  PUT "The length of the string is: " result2;
  PUT "The length of the string is: " result3;
  PUT "The length of the string is: " result4;
RUN;
```

This produces the following output:

```
The length of the string is: 51
The length of the string is: 4
The length of the string is: 1
The length of the string is: 12
```

- The first use of the function returns the value 51, which is the length of the string including trailing spaces.
- The second use of the function returns the value 4, which is the number of (trailing) spaces in the string.
- The third use of the function returns the value 1, because the string is a null.
- The fourth use of the function returns the value 12, because the string contains a numeric value. A message is written to the log noting that the numeric values have been converted to character values.

Example – strings in variables of a specified length

In this example, the function is used to find the length of various strings that have been assigned to variables with a specified length. The result is written to the log.

```
DATA _NULL_;
  twslt = "Bicycle";
  LENGTH twsl twsln twslt $10;
  twsl = "Bicycle";
  result = LENGTHC(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsl="Twenty-three bicycles";
  result = LENGTHC(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsln="";
  result = LENGTHC(twsln);
  ps = QUOTE(twsln);
  PUT "The string is now: " ps " and its length is: " result;

  result = LENGTHC(twslt);
  ps = QUOTE(twslt);
  PUT "The string is now: " ps " and its length is: " result;
RUN;
```

This produces the following output:

```
The string is now: "Bicycle  " and its length is: 10
The string is now: "Twenty-thr" and its length is: 10
The string is now: " " and its length is: 10
The string is now: "Bicycle" and its length is: 7
```

Strings shorter than the variable length have been padded with spaces, and strings longer than the variable length have been truncated. In this example, therefore, the result returned is 10 for each string, because both variables have been assigned a length of ten characters and padding spaces are not ignored.

The variable `twslt` is set before its length is specified in the `LENGTH` statement; therefore its length is returned as 7, the number of characters in the string.

Contrast this result to that obtained with [LENGTH](#) (page 2061), [LENGTHM](#) (page 2065) and [LENGTHN](#) (page 2067).

LENGTHM

Returns the length of a string, including any leading or trailing spaces. If the string is null 1 is returned.

```
➤ LENGTHM ➤ ( string ) ➤
```

If a number is specified as an argument, a value of 12 is returned, and a message written to the log noting that numeric values have been converted to character values.

Return type: Numeric

string

Type: Character

The string to be examined.

If the length of a variable has been specified, and the string to be checked is contained in that variable, then the number returned will correspond to the length of the variable.

Basic example

In this example, the function is used to return results for various strings. The result is written to the log.

```
DATA _NULL_;
  number = 1234567123456789;
  result1 = LENGTHM('    Magnificent Bike Company Bike A 1 2 50    London    ');
  result2 = LENGTHM('    ');
  result3 = LENGTHM('');
  result4 = LENGTHM(number);

  PUT "The number of bytes occupied by the string is: " result1;
  PUT "The number of bytes occupied by the string is: " result2;
  PUT "The number of bytes occupied by the string is: " result3;
  PUT "The number of bytes occupied by the string is: " result4;
RUN;
```

This produces the following output:

```
The number of bytes occupied by the string is: 55
The number of bytes occupied by the string is: 4
The number of bytes occupied by the string is: 1
The number of bytes occupied by the string is: 12
```

In this example:

- The first use of the function returns the value 55, which is the number of bytes of memory occupied by the string; the string in this case includes leading and trailing spaces.
- The second use of the function returns the value 4, which is the number of bytes of memory occupied by the string.
- The third use of the function returns the value 1, as the string contains a null.
- The fourth use of the function returns the value 12, because the string contains a numeric value. A message is written to the log noting that the numeric values have been converted to character values.

Example – strings in variables of a specified length

In this example, the function is used to return results for various strings that have been assigned to variables with a specified length. The result is written to the log.

```
DATA _NULL_;
  twslt = "Bicycle";
  LENGTH twsl twsln twslt $10;
  twsl = "Bicycle";
  result = LENGTHM(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsl="Twenty-three bicycles";
  result = LENGTHM(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsln="";
  result = LENGTHM(twsln);
  ps = QUOTE(twsln);
  PUT "The string is now: " ps " and its length is: " result;

  result = LENGTHM(twslt);
  ps = QUOTE(twslt);
  PUT "The string is now: " ps " and its length is: " result;

RUN;
```

This produces the following output:

```
The string is now: "Bicycle  " and its length is: 10
The string is now: "Twenty-thr" and its length is: 10
The string is now: " " and its length is: 10
The string is now: "Bicycle" and its length is: 7
```

Strings shorter than the variable length have been padded with spaces, and strings longer than the variable length have been truncated. In this example, therefore, the result returned is 10 for each string, because both variables have been assigned a length of ten characters and padding spaces are not ignored.

The variable `twslt` is set before its length is specified in the `LENGTH` statement; therefore its length is returned as 7, the number of characters in the string.

Contrast this result to that obtained with [LENGTH](#) (page 2061), [LENGTHC](#) (page 2063) and [LENGTHN](#) (page 2067).

LENGTHN

Returns the length of a character string, including leading spaces but excluding trailing spaces, and returns 0 (zero) if the string is null (' ') or contains only spaces.

➤ **LENGTHN** ➤ (*string*) ➤

If a number is specified as an argument, a value of 12 is returned, and a message written to the log noting that numeric values have been converted to character values.

Return type: Numeric

string

Type: Character

The string for which the length is required.

If the length of a variable has been specified, and the string to be checked is contained in that variable, then the length returned will be length of the variable minus any leading spaces. See below for an example.

Basic example

In this example, the function is used to find the length of various strings. The result is written to the log.

```
DATA _NULL_;
  number = 1234567123456789;
  result1 = LENGTHN('    Magnificent Bike Company Bike A 1 2 50    London  ');
  result2 = LENGTHN('    ');
  result3 = LENGTHN('');
  result4 = LENGTHN(number);

  PUT "The length of the string is: " result1;
  PUT "The length of the string is: " result2;
  PUT "The length of the string is: " result3;
  PUT "The length of the string is: " result4;
RUN;
```

This produces the following output:

```
The length of the string is: 51
The length of the string is: 0
The length of the string is: 0
The length of the string is: 12
```

In this example:

- The first use of the function returns the value 51, which is the length of the string including leading spaces but excluding trailing spaces.
- The second use of the function returns the value 0, as the string consists only of spaces.
- The third use of the function returns the value 0, as the string consists of a null.
- The fourth use of the function returns the value 12, because the string contains a numeric value. A message is written to the log noting that the numeric values have been converted to character values.

Example – strings in variables of a specified length

In this example, the function is used to find the length of various strings that have been assigned to variables with a specified length. The result is written to the log.

```
DATA _NULL_;
  LENGTH twsl twsln $15;
  twsl="Bicycle";
  result = LENGTHN(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsl="   Bicycle   ";
  result = LENGTHN(twsl);
  ps = QUOTE(twsl);
  PUT "The string is now: " ps " and its length is: " result;

  twsln="";
  result = LENGTHN(twsln);
  ps = QUOTE(twsln);
  PUT "The string is now: " ps " and its length is: " result;
RUN;
```

This produces the following output:

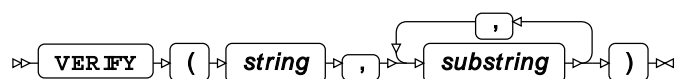
```
The string is now: "Bicycle   " and its length is: 7
The string is now: "   Bicycle" and its length is: 10
The string is now: "          " and its length is: 0
```

This function includes leading spaces from the count, but ignores trailing spaces, including the padding spaces applied to the strings to make them match the length specified for the variable. However, the string that is longer than the specified variable length has been truncated, and the length returned by the function is the same as the variable length.

Contrast this result to that obtained with [LENGTH](#) (page 2061), [LENGTHC](#) (page 2063) and [LENGTHM](#) (page 2065).

VERIFY

Returns the position of the first character of the source string that does not match a specified substring.



If the source string contains none of the specified substrings, 0 (zero) is returned.

Return type: Numeric

string

Type: Character

The string to be searched for specified substrings.

substring

Type: Character

A substring to be found in *source*.

Example – verifying string contains single characters

In each of the following examples, the result is written to the log.

In this example, the function searches for the character a.

```
DATA _NULL_;  
    result = VERIFY('aaabbb', 'a');  
    PUT 'The character is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 4
```

In this example, the function searches for the character b.

```
DATA _NULL_;  
    result = VERIFY('aaabbb', 'b');  
    PUT 'The character is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 1
```

In this example, the function searches for the characters a and b .

```
DATA _NULL_;  
    result1 = VERIFY('aaabbb', 'a','b');  
    PUT 'The character is first NOT found at position: ' result1;  
RUN;
```

This produces the following output:

```
The character is first NOT found at position: 0
```

The function returns 0 (zero) in this instance as there is nowhere in the string where one of the characters does not exist.

Example – verifying string contains substrings

In each of the following examples, the result is written to the log.

In this example, the function searches for the string BigCat.

```
DATA _NULL_;  
    result = VERIFY('BigCatLeopard', 'BigCat');  
    PUT 'The string is first NOT found at position: ' result;  
RUN;
```

This produces the following output:

```
The string is first NOT found at position: 7
```

In this example, the function searches for the strings `Big` and `Cat`.

```
DATA _NULL_;
  result = VERIFY('BigCatLeopard', 'Big', 'Cat');
  PUT 'The strings are first NOT found at position: ' result;
RUN;
```

This produces the following output:

```
The strings are first NOT found at position: 7
```

Character position seven is the first at which neither of the strings `Big` and `Cat` is found.

In this example, the function searches for the strings `Big`, `Cat` and `Leopard`. The result is written to the log.

```
DATA _NULL_;
  result = VERIFY('BigCatLeopard', 'Big', 'Cat', 'Leopard');
  PUT 'The strings are first NOT found at position: ' result;
RUN;
```

This produces the following output:

```
The strings are first NOT found at position: 0
```

The function returns 0 as there is no position in the source strings at which one of the strings is not found.

Finally, in the following example the function searches for the string `Leopard`:

```
DATA _NULL_;
  result = VERIFY('BigCatLeopard', 'Leopard');
  PUT 'The strings are first NOT found at position: ' result;
RUN;
```

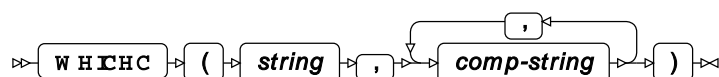
This produces the following output:

```
The string is first NOT found at position: 1
```

The function returns 1 because that is the first position at which `Leopard` is not found.

WHICHC

Returns the position of the first string in a list of strings that matches a specified string.



Identifies the first string in a list of strings that matches the string you specify, and returns as a number the ordinal position of the matched string in the list of strings. The strings for which to search are specified as one or more comma-separated strings. For example, if the list of strings has four items, and the third string matches, then 3 is returned. If no matching string is found, 0 is returned.

Return type: Numeric

string

Type: Character

The string you want to find in a list of other strings.

comp-string

Type: Character

A string to be matched with *string*.

Example

In this example, the function is used to find the string `Bike` in a list of strings. The result is written to the log.

```
DATA _NULL_;
  result1 = WHICHC('Bike','Bice', 'Boke', 'Bike');
  PUT "The string is number " result1 "in the list";
RUN;
```

This produces the following output:

```
The string is number 3 in the list
```

Modifying strings, characters and numerics

Return a source string modified in a particular way.

A specified string can be modified in various ways, depending on the function selected. You can add quotes to or remove them from strings, repeat the string, convert specified characters to other characters, and so on.

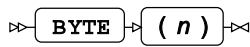
BYTE ↗	2074
Returns the result of converting a specified decimal value to a character. You might need to do this if, for example, data contains decimal values for characters rather than the characters themselves.	
COMPRESS ↗	2075
Returns the string that is the result of removing specified characters from a source string. This function can be used to remove spaces, separators, tabs, numbers, and so on, depending on modifiers you specify in the function.	

DEQUOTE ↗	2078
Returns the string that results from stripping quotation marks from a specified string.	
KCOMPRESS ↗	2079
Returns the string that is the result of removing specified characters from a source string comprised of DBCS characters.	
KCVT ↗	2081
Returns the string that results from converting a source string encoded in one character set to another character set.	
KREVERSE ↗	2082
Returns a string that is the reverse of the specified string of DBCS characters.	
KTRANSLATE ↗	2083
Returns the string that results from changing a specified character in a DBCS string.	
KTRUNCATE ↗	2084
Returns the string that results from truncating a specified DBCS string at a specified position.	
KUPDATE ↗	2085
Returns the string that results from replacing a substring at a specified position in a DBCS string.	
KUPDATEB ↗	2086
Return the string that results from replacing a substring at a specified byte position in a DBCS string.	
MISSING ↗	2087
Returns a flag that indicates whether a string is missing a value.	
QUOTE ↗	2088
Returns the string that results from wrapping a specified string in quotation marks.	
REPEAT ↗	2089
Returns the string that results from repeating the specified character or string a specified number of times.	
REVERSE ↗	2090
Returns the string that results from reversing a specified string.	
SOUNDEX ↗	2090
Returns the soundex equivalent of a specified string.	
TRANSLATE ↗	2092
Returns a string that consists of a source string that has had all instances of a specified string translated to another specified character. You can change more than one character.	
TRANSTRN ↗	2093
Replace a substring in a source string with another substring.	
TRANTAB ↗	2094
Returns strings modified by translation tables.	
TRANWRD ↗	2096
Replace a word in a source string with another word.	

CALL MISSING ↗	2097
Assigns numeric missing values to numeric values, and spaces to string values. By default, numeric missing values are represented by a full stop (.). String values are replaced with the same number of spaces as there are characters in the original string.	
CALL SORTC ↗	2098
Sorts a list of strings.	

BYTE

Returns the result of converting a specified decimal value to a character. You might need to do this if, for example, data contains decimal values for characters rather than the characters themselves.



Return type: Character

n

Type: Numeric

An unsigned decimal number. If *n* is not a decimal number, an error occurs.

Example

In this example, the SUBSTR function is used to search the string for a number at the specified location, and returns the result to NUM. The BYTE function is then used to convert that number. The result is written to the log.

```

DATA _NULL_;
  num=SUBSTR("101 01adx", 1,3);
  result = BYTE(num);
  PUT "Converted value is: " result;
RUN;
  
```

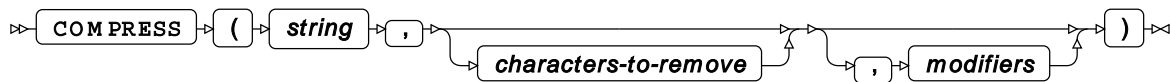
This produces the following output:

```
Converted value is: e
```

The function converts the value found in num, which is 101, to a character, which will be e in the US English ASCII code page.

COMPRESS

Returns the string that is the result of removing specified characters from a source string. This function can be used to remove spaces, separators, tabs, numbers, and so on, depending on modifiers you specify in the function.



Return type: Character

string

Type: Character

The string to be examined.

characters-to-remove

Optional argument

Type: Character

A list of characters to be removed from the string. For example, setting this argument to 'aeiou' would remove all vowels from the string. If you omit this argument, only those characters specified by *modifiers* are removed. If you set this argument to null (' '), all spaces (but not tabs) are removed. If you omit this argument but want to specify the *modifiers* argument, you must insert the associated separator.

modifiers

Optional argument

One or more characters that specify how the output should be modified. You can specify more than one modifier. For example, you can specify 'C', or 'CAS'. The following modifiers are available:

"A"

All alphabetic characters will be removed

"C"

All control characters are removed.

"D"

All decimal numbers (0 through 9) are removed.

"F"

All characters considered a valid character for the first position in a variable name will be removed.

"G"

All graphics characters are removed.

"H"

All spaces including tabs are removed.

"I"

Ignore case of characters to remove. For example, if you specify `B` in *characters-to-remove*, and then specify this modifier, both `B` and `b` will be removed.

"K"

Causes modifiers to act in the opposite way. So, for example, if `L` is set, instead of removing all lower-case characters, the function will instead remove all uppercase characters.

"L"

Removes all lower-case alphabetic characters.

"N"

A character considered a valid character for a variable name will be considered a delimiter.

"O"

Store the list of characters to be removed the first time the function is called. This can increase processing efficiency if there are multiple calls to the function.

"P"

Removes all punctuation marks.

"S"

Removes all spaces, including tabs.

"T"

If you specify *characters-to-remove* other than spaces (the default), any trailing spaces at the end of the value specified for *string* are retained. To remove these spaces, use this modifier. For example:

```
DATA _NULL_;  
    result = QUOTE(compress('London    ', 'L', 'T'));  
    PUT result;  
RUN;
```

returns `"ondon"`; without the modifier, `"ondon "` would have been returned.

"U"

Removes all upper-case alphabetic characters.

"W"

Removes all printing characters.

"X"

Removes all strings/characters that comprise hexadecimal numbers (0-9, a-f and A-F).

Example – removing spaces

In this example, the function is used to remove spaces. The result is written to the log.

```
DATA _NULL_;  
  result = COMPRESS('Steve Bike Company A 1 2 50 London', ' ');  
  PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: SteveBikeCompany A 1250 London
```

Spaces have been removed, but tabs have been retained.

Example – removing spaces and hexadecimal numbers

In this example, the function is used to compress multiple spaces to nulls, and to remove characters that might represent hexadecimal numbers. The result is written to the log.

```
DATA _NULL_;  
  result = COMPRESS('Steve Bike Company A 1 2 50 London', ' ', 'X');  
  PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: Stvikompny London
```

The function has removed spaces (but retained tabs) and characters that might represent hexadecimal numbers.

Example – upper-case characters

In this example, the function is used to compress one or more spaces to nulls, and to remove upper-case characters. This produces the following output:

```
DATA _NULL_;  
  result = COMPRESS('Steve Bike Company A 1 2 50 London',,, 'U');  
  PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: teve ike ompany 1 2 50 ondon
```

All upper-case characters have been removed. No other characters were removed, because the second argument (*characters-to-remove*) is not specified.

Example – removing specified alphabetic and upper-case characters

In this example, the function is used to remove lower-case letter *e* and upper-case characters. This produces the following output:

```
DATA _NULL_;  
    result = COMPRESS('Steve Bike Company   A   1 2 50   London','e', 'U');  
    PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: tv ik ompany       1 2 50   ondon
```

All occurrences of the letter *e* have been removed, and all upper-case characters as specified by the *U* option.

Example – removing multiple specified characters

In this example, the function is used to remove all vowels and all numeric characters. The result is written to the log.

```
DATA _NULL_;  
    result = COMPRESS('Steve Bike Company   A   1 2 50   London','aeio', 'D');  
    PUT "The string returned is: " result;  
RUN;
```

This produces the following output:

```
The string returned is: Stv Bk Cmpny   A       Lndn
```

The letters *a*, *e*, *i* and *o* have been removed. All numeric characters have also been removed, as specified by the *D* option.

DEQUOTE

Returns the string that results from stripping quotation marks from a specified string.

➤ DEQUOTE ➡ (*string*) ➤

Return type: Character

string

Type: Character

The string to be stripped.

Example

In this example, the function is used to strip the opening and closing quotation marks from the source string to the function. The result is written to the log.

```
DATA _NULL_;  
    result = DEQUOTE('Magnificent Bike Company A 1 2 50    London');  
    PUT result;  
RUN;
```

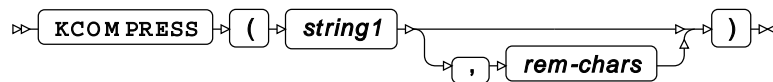
This produces the following output:

```
Magnificent Bike Company A 1 2 50    London
```

This is the original string with the quotation marks removed.

KCOMPRESS

Returns the string that is the result of removing specified characters from a source string comprised of DBCS characters.



Removes occurrences of specified characters from a string consisting of characters from a double-byte character set, and returns the modified string. This function can be used to remove one or more characters you specify.

Return type: Character

string1

Type: Character

The string to be examined.

rem-chars

Optional argument

Type: Character

A list of characters or a variable containing a list of characters to be removed from *source*. For example, setting this argument to '青柳鼠' would remove all occurrences of those characters from *source*. If you omit this argument, all spaces and tabs are removed. If you set it to null (' '), all spaces are removed, but tabs are retained.

Basic example

In this example, the function is used to compress spaces to nulls. The result is written to the log. In this example there is a tab between 猩々緋 and 青柳鼠.

```
DATA _NULL_;  
  result = KCOMPRESS('青碧 熨斗目花色 ときがら茶 猩々緋 青柳鼠');  
  PUT result;  
RUN;
```

This produces the following output:

```
青碧熨斗目花色ときがら茶猩々緋青柳鼠
```

Spaces and tabs have been removed.

Example – removing spaces while retaining tabs

In this example, the function is used to compress multiple spaces, excluding tabs. The result is written to the log. In this example there is a tab between 猩々緋 and 青柳鼠.

```
DATA _NULL_;  
  result = KCOMPRESS('青碧 熨斗目花色 ときがら茶 猩々緋 青柳鼠', ' ');  
  PUT result;  
RUN;
```

This produces the following output:

```
青碧熨斗目花色ときがら茶猩々緋青柳鼠
```

Spaces and tabs have been removed.

Example – removing spaces while retaining tabs

In this example, the function is used to compress multiple spaces, excluding tabs. The result is written to the log. In this example there is a tab between 猩々緋 and CALL ALLPERM>青柳鼠.

```
DATA _NULL_;  
  result = KCOMPRESS('青碧 熨斗目花色 ときがら茶 猩々緋 青柳鼠', ' ');  
  PUT result;  
RUN;
```

This produces the following output:

```
青碧熨斗目花色ときがら茶猩々緋 青柳鼠
```

All spaces have been removed, excluding tabs

Example – removing other characters

In this example, the function is used to remove other characters. The result is written to the log.

```
DATA _NULL_;  
  result = KCOMPRESS('青碧、熨斗目花色、ときがら茶:とうきん煤竹 空五倍子色', '\、:');  
  PUT result;  
RUN;
```

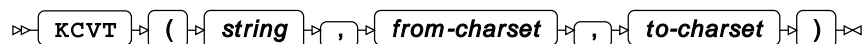
This produces the following output:

```
青碧熨斗目花色ときがら茶とうきん煤竹 空五倍子色
```

The commas and colon have been removed, but the space retained.

KCVT

Returns the string that results from converting a source string encoded in one character set to another character set.



Return type: Character

Converts a source string from one character set to another character set, and returns the resulting string. You must specify both the character set to be converted from and converted to.

string

Type: Character

The string to be converted.

from-charset

Type: Character

The character set of the source string.

to-charset

Type: Character

The character set to which the string will be converted.

You will need to know the character set of the source string to convert it. If you specify an unrecognised character set for *from-charset* or *to-charset*, the function fails and an error message is written to the log. If a character in the string cannot be mapped to another character in the *to-charset*, the message `WARNING: An unmappable character was encountered during character transcoding` is written to the log.

Example

In this example, the function is used to convert a string from the windows-1252 dataset to UTF-8. The result is written to the log.

```
DATA _NULL_;  
    result = QUOTE(KCVT('†€', 'windows-1252', 'UTF-8'));  
    PUT result;  
RUN;
```

This produces the following output:

```
"â€ â€;â,¬"
```

The quotation marks are added by the `QUOTE` function. The UTF-8 conversion cannot be rendered correctly in WPS on Windows using the default character set.

KREVERSE

Returns a string that is the reverse of the specified string of DBCS characters.

⇒ **KREVERSE** (*string*) ⇒

Reverses a string containing characters from a double-byte character set (DBCS), and returns the result.

Return type: Character

string

Type: Character

The string to be reversed.

Example

In this example, a string is reversed. The result is written to the log.

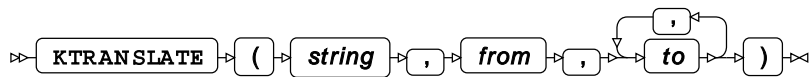
```
DATA _NULL_;  
    sentence='熨斗目花色';  
    result=KREVERSE(sentence);  
    PUT 'The word reversed is: result';  
RUN;
```

This produces the following output:

```
色花目斗熨
```


KTRANSLATE

Returns the string that results from changing a specified character in a DBCS string.



Finds all instance of a specified character in a string consisting of double-byte character set (DBCS) characters, changes (translates) them to another specified character, and returns the modified string. You can change more than one character.

Return type: Character

string

Type: Character

The string to be modified.

from

Type: Character

The character to convert (translate) to.

to

Type: Character

The character to convert (translate) from.

Note:

You can also specify multiple characters to change as groups of *to/from* arguments; for example, `KTRANSLATE(orig, '鼠青', '柳斗')`

Example

In this example, the function is used to convert the characters 柳 and 斗 into 鼠 and 青. The result is written to the log.

```
DATA _NULL_;
  orig = "青碧 熨斗目花色 猩々緋 青柳鼠 ときから茶";
  result1 = KTRANSLATE(orig, '鼠', '青', '柳', '斗');
  result2 = KTRANSLATE(orig, '鼠柳', '青斗');
  PUT "Original string is: " orig;
  PUT "Transformed string is: " result1;
  PUT "Transformed string is: " result2;
RUN;
```

This produces the following output:

```
Original string is: 青碧 熨斗目花色 猩々緋 青柳鼠 ときがら茶
Transformed string is: 鼠碧 熨柳目花色 猩々緋 鼠柳鼠 ときがら茶
Transformed string is: 鼠碧 熨柳目花色 猩々緋 鼠柳鼠 ときがら茶
```

In the first translation, multiple characters to be translated have been grouped in the `to` and `from` arguments. In the second translation, each character to translate to and from have been specified in separate `to` and `from` arguments. The results are, however, the same.

KTRUNCATE

Returns the string that results from truncating a specified DBCS string at a specified position.

➤ **KTRUNCATE** ➤ (➤ *string* ➤ , ➤ *length* ➤) ➤

Truncates a string consisting of characters from a double-byte character set (DBCS) at a specified position, and returns the truncated string.

Return type: Character

string

Type: Character

The string to be truncated.

length

Type: Numeric

The number of bytes of the string in *argument*, starting from the first character, to be retained.

Example

In this example, the function is used to truncate a string. The result is written to the log.

```
DATA _NULL_;
  result1 = KTRUNCATE('青碧 熨斗目花色 猩々緋 青柳鼠', 12);
  PUT "The truncated string is: " result1;
RUN;
```

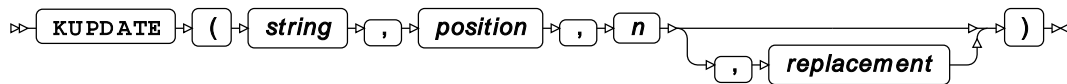
This produces the following output:

```
The truncated string is: 青碧 熨
```

The resulting string retains the first four characters (including the space), which is the first twelve bytes of the source string.

KUPDATE

Returns the string that results from replacing a substring at a specified position in a DBCS string.



Replaces a substring in a string consisting of characters from a double-byte character set (DBCS), with another string. The replacement starts at a specified point in the string, and replaces a specified number of characters. The number of characters replaced do not have to match the length of the replacement string.

Return type: Character

string

Type: Character

The source string.

position

Type: Numeric

The character position in *source* at which insert *replacement*. If this is beyond the end of *source*, *replacement* is appended to *source*.

n

Type: Character or numeric value

The number of characters in *source* to be replaced.

replacement

Optional argument

Type: Character

The string to be inserted into *source*.

Example

In this example, the function is used to insert the string テレビ starting at the fourteenth character in the source string. The result is written to the log.

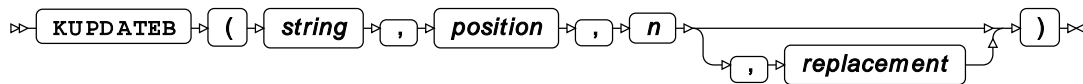
```
DATA _NULL_;
  result = KUPDATE("青碧 熨斗目花色 猩々緋 青柳鼠 ときがら茶", 14, 3, 'テレビ');
  PUT "Modified output: " result;
RUN;
```

In this example,

Modified output: 青碧 熨斗目花色 猩々緋 テレビ ときがら茶

KUPDATEB

Return the string that results from replacing a substring at a specified byte position in a DBCS string.



Replaces a substring in a string consisting of characters from a double-byte character set with another string. The replacement starts at a specified byte in the string, and optionally replaces a specified number of characters. The number of characters replaced do not have to match the length of the replacement string.

Return type: Character

string

Type: Character

The source string.

position

Type: Numeric

The byte in *source* at which to insert *replacement*. If this is beyond the end of *source*, *replacement* is appended to *source*.

n

Type: Character or numeric value

The number of bytes in *source* to be replaced.

replacement

Optional argument

Type: Character

The string to be inserted into *source*.

Example

In this example, the function is used to insert the string テレビ starting at the 34th byte in the string. The result is written to the log.

```
data example;
  result = KUPDATEB("青碧 熨斗目花色 猩々緋 青柳鼠 ときから茶", 34, 10, 'テレビ');
  PUT "Modified output: " result;
RUN;
```

This produces the following output:

```
Modified output: 青碧 熨斗目花色 猩々緋 テレビ ときから茶
```

MISSING

Returns a flag that indicates whether a string is missing a value.

➤ **MISSING** ➤ (*argument*) ➤

Examines the argument provided, which can be a number, string or variable, and flags whether it is, or contains, something other than null, only spaces or a missing value. You can use this function, for example, to ensure an observation or variable contains meaningful data.

Return type: Numeric

If a string contains null (' '), only spaces, or a missing value, then 1 is returned. If a string contains any other characters, then 0 is returned.

argument

Type: Character or numeric value

The string to be examined.

Example

In this example, the function is used to examine various strings. The result is written to the log.

```
DATA _NULL_;  
  num=num2-3;  
  result1 = MISSING('');  
  result2 = MISSING('   ');  
  result3 = MISSING(' h ');  
  result4 = MISSING(5);  
  result5 = MISSING(num);  
  
  PUT result1;  
  PUT result2;  
  PUT result3;  
  PUT result4;  
  PUT result5;  
RUN;
```

This produces the following output:

```
1  
1  
0  
0  
1
```

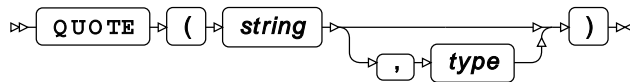
In this example:

- `result1` is 1, as the first string is null.
- `result2` is 1, as the second string consists entirely of spaces.
- `result3` is 0, as the string contains a character that is not null or space.

- `result4` is 0, as the value is numeric.
- `result5` is 1, as the value is missing because the uninitialised variable `num2` was used in the expression.

QUOTE

Returns the string that results from wrapping a specified string in quotation marks.



By default, double quotation marks are added to the string. You can specify single quotation marks if required.

Return type: Character

string

Type: Character

The string to which quotation marks will be applied.

type

Optional argument

Type: Character

Specifies the type of quotation mark. Enter " ' " for single quotation marks.

Example – using default quotation marks

In this example, the string is wrapped in default double quotation marks. The result is written to the log.

```
DATA _NULL_;
  sentence='London Bikes';
  result=QUOTE(sentence);
  PUT "Quoted string: " result;
RUN;
```

This produces the following output:

```
Quoted string: "London Bikes"
```

Example – using single quotation marks

In this example, the string is wrapped in single double quotation marks. The result is written to the log.

```
DATA _NULL_;  
  sentence='London Bikes';  
  result=quote(sentence, '"');  
  PUT "Quoted string: " result;  
RUN;
```

This produces the following output:

```
Quoted string: 'London Bikes'
```

REPEAT

Returns the string that results from repeating the specified character or string a specified number of times.

➤ **REPEAT** ➤ (*string* , *n*) ➤

Return type: Character

string

Type: Character

A character or a string to be repeated.

n

Type: Numeric

An integer that specifies the number of repetitions.

The result returned is a string containing the specified number *n* of instances of the character or string, plus the initial character or string. Therefore, if *n* is 2, for example, there will be three instances of the character or string. The characters or strings returned are concatenated with the original. See the example below for more detail.

Example

In this example, a string is repeated three times. The result is written to the log.

```
DATA _NULL_;  
  sentence='London Calling';  
  result=REPEAT(sentence, 3);  
  PUT "Resulting string is: " result;  
RUN;
```

This produces the following output:

```
Resulting string is: London CallingLondon CallingLondon CallingLondon Calling
```

There are three repeated strings concatenated with the original string, giving a total of four instances of the original string.

REVERSE

Returns the string that results from reversing a specified string.

➤ **REVERSE** ➤ (*string*) ➤

Return type: Character

string

Type: Character

The string to be reversed.

Example

In this example, a string is reversed. The result is written to the log.

```
DATA _NULL_;  
  sentence='London Calling';  
  result=REVERSE(sentence);  
  PUT "Reversed string is: " result;  
RUN;
```

This produces the following output:

```
Reversed string is: gnillaC nodnoL
```

SOUNDEX

Returns the soundex equivalent of a specified string.

➤ **SOUNDEX** ➤ (*string*) ➤

Soundex is an algorithm that enables words that sound alike to be represented in a similar way, despite differences in spelling. For example, Smith, Smyth and Smythe can all be represented by the soundex value, S53.

The soundex algorithm retains the first letter of a word (converting it to upper case, if necessary), drops all other occurrences of the letters AEIOUYHW, and then assigns a numeric value to other letters. If there are two or more adjacent letters with the same number only the first number is retained; letters with the same number separated by H or W are coded as a single number; letters with the same number separated by a vowel are coded twice. For more information on the soundex algorithm, see the soundex coding rules [🔗](#) at the *National Archives and Records Administration Web* site.

Note:

The soundex algorithm truncates the result at four characters, and pads results shorter than four characters with zeros. The `SOUNDEX` function returns all characters formed by the algorithm through to the end of the string, and does not pad returned values shorter than four characters with zeros; if you need to make the value returned by `SOUNDEX` match the soundex algorithm, you will have to perform further operations, such as using `SUBSTR` to return only the first four characters.

Return type: Character

string

Type: Character

The source string.

Example

In this example, the function is used to convert various strings to soundex form. The result is written to the log.

```
DATA _NULL_;
  result1 = SOUNDEX('Bike');
  result2 = SOUNDEX('Boke');
  result3 = SOUNDEX('Orange');
  result4 = SOUNDEX('Oranj');
  result5 = SOUNDEX('Orange Juice');

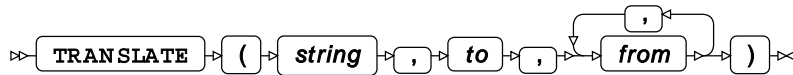
  PUT result1;
  PUT result2;
  PUT result3;
  PUT result4;
RUN;
```

This produces the following output:

```
B2
B2
O652
O652
O65222
```

TRANSLATE

Returns a string that consists of a source string that has had all instances of a specified string translated to another specified character. You can change more than one character.



Return type: Character

string

Type: Character

The string in which character changes are required.

to

Type: Character

One or more characters to convert (translate) to.

from

Type: Character

One or more characters to convert (translate) from.

Note:

You can also specify multiple characters to change as pairs of to/from arguments; for example:

```
translate("Bike Company", 'a', 'o', 'u', 'i')
```

Example

In this example, the function is used to convert the letters `a` and `i` into `o` and `u`. The result is written to the log.

```
DATA _NULL_;  
  result = TRANSLATE("Magnificent Bike Company", 'ou', 'ai');  
  PUT result;  
RUN;
```

This produces the following output:

```
Mognufucent Buke Compony
```

TRANSTRN

Replace a substring in a source string with another substring.

➤ **TRANSTRN** ➤ (*string* , *from* , *to*) ➤

Finds a specified substring in a source string, changes (translates) all instances of it to a specified substring, and returns the modified source string.

Return type: Character

string

Type: Character

The string which contains one or more substrings to be changed.

from

Type: Character

The substring to replace.

to

Type: Character

The replacement substring. A null (zero length string) can be used as the replacement string.

Note:

The only difference between `TRANWRD` and `TRANSTRN` is in how they handle a null (empty) string specified in the *to* argument. With `TRANWRD`, if *to* is null a space is used instead.

The string provided in *from* must match a substring in the source string exactly for a replacement to be made.

Example

In this example, the substring `Bike` is replaced with `Car`. The result is written to the log.

```
DATA _NULL_;
  result1=transtrn('Magnificent Bike Company', 'Bike', 'Car');
  result2=transtrn('Magnificent Bike Company', 'Bike', trimn(''));
  PUT result1;
  PUT result2;
RUN;
```

The first use of the function returns the result:

```
Magnificent Car Company
```

Note:

If you had specified `bike` as the *from* string, no changes would have been made, as `bike` would not match `Bike`.

The second use of the function returns the result:

Magnificent Company

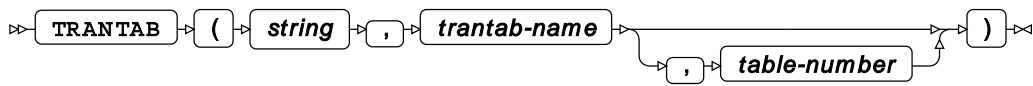
In this case, `Bike` has been replaced with the null string (`' '`).

Note:

The example uses `TRIMN` to ensure any spaces are removed from the null string, as null strings are expanded to a space in the `DATA` step.

TRANTAB

Returns strings modified by translation tables.



Return type: Character

string

Type: Character

The string to be modified

trantab-name

Type: Character

The name of a catalog containing the translation table.

table-number

Optional argument

Type: Numeric

The number of a specific table within catalogue.

Example

In this example, the function is used to modify a string using tables. The tables are first set up using PROC TRANTAB and PROC CATALOG. The TRANTAB function is then used to modify a string. The result is written to the log.

```
/* Create an example TRANTAB.
   Table 1 decrements every byte.
   Table 2 increments every byte.
*/
PROC TRANTAB table=WPSTEST;
REPLACE 0
  'FF000102030405060708090A0B0C0D0E'x
  '0F101112131415161718191A1B1C1D1E'x
  '1F202122232425262728292A2B2C2D2E'x
  '2F303132333435363738393A3B3C3D3E'x
  '3F404142434445464748494A4B4C4D4E'x
  '4F505152535455565758595A5B5C5D5E'x
  '5F606162636465666768696A6B6C6D6E'x
  '6F707172737475767778797A7B7C7D7E'x
  '7F808182838485868788898A8B8C8D8E'x
  '8F909192939495969798999A9B9C9D9E'x
  '9FA0A1A2A3A4A5A6A7A8A9AAABACADAE'x
  'AFB0B1B2B3B4B5B6B7B8B9BABBBBCBDBE'x
  'BFC0C1C2C3C4C5C6C7C8C9CACBCCCDCE'x
  'CFD0D1D2D3D4D5D6D7D8D9DADBDCDDDE'x
  'DFE0E1E2E3E4E5E6E7E8E9EAEBECEDEE'x
  'EFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFE'x;
SWAP;
REPLACE 0
  '0102030405060708090A0B0C0D0E0F10'x
  '1112131415161718191A1B1C1D1E1F20'x
  '2122232425262728292A2B2C2D2E2F30'x
  '3132333435363738393A3B3C3D3E3F40'x
  '4142434445464748494A4B4C4D4E4F50'x
  '5152535455565758595A5B5C5D5E5F60'x
  '6162636465666768696A6B6C6D6E6F70'x
  '7172737475767778797A7B7C7D7E7F80'x
  '8182838485868788898A8B8C8D8E8F90'x
  '9192939495969798999A9B9C9D9E9FA0'x
  'A1A2A3A4A5A6A7A8A9AAABACADAEAFB0'x
  'B1B2B3B4B5B6B7B8B9BABBBBCBDBEBFC0'x
  'C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0'x
  'D1D2D3D4D5D6D7D8D9DADBDCDDDEDFE0'x
  'E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0'x
  'F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF00'x;
SWAP;
SAVE BOTH;
RUN;

/* Modify the catalog description of the new trantab */
PROC CATALOG c=wpsuser.profile entrytype=trantab;
MODIFY WPSTEST(description='Decrement/Increment order');
RUN;

/* Modify a string using the tables of the trantab. */
DATA _NULL_;
  s = '123456789BCDEFGbcdefg';
  s1 = TRANTAB(s, 'WPSTEST', 1);
```

```
s2 = TRANTAB(s, 'WPSTEST', 2);  
s3 = TRANTAB(s1, 'WPSTEST', 2);  
PUT "All characters decreased one step from original: " s1;  
PUT "All characters increased one step from original: " s2;  
PUT "Modified string decreased one step (back to original): " s3; ;  
RUN;
```

This produces the following output:

```
All characters decreased one step from original: 012345678ABCDEFabcdef  
All characters increased one step from original: 23456789:CDEFGHcdefgh  
Modified string decreased one step (back to original): 123456789BCDEFGbcdefg
```

TRANWRD

Replace a word in a source string with another word.

➤ **TRANWRD** ➤ (*string* , *from* , *to*) ➤

Finds a specified word in a source string, changes all instances of it to a specified word, and returns the modified source string.

Note:

The only difference between `TRANWRD` and `TRANSTRN` is in how they handle a null (empty) string specified in the `to` argument. With `TRANSTRN`, `to` can be zero length (null).

Return type: Character

string

Type: Character

The string which contains one or more word to be changed.

from

Type: Character

The word to replace.

to

Type: Character

The replacement word. If this argument is null, a space is used as the replacement string.

The word provided in *from* must match a substring in the source string exactly for a replacement to be made.

Example

In this example, the word `Bike` is replaced with other strings. The result is written to the log.

```
DATA _NULL_;  
  a='';  
  result1=tranwrd('Magnificent Bike Company', 'Bike', 'Car');  
  result2=tranwrd('Magnificent Bike Company', 'Bike', trimn(a));  
  PUT result1;  
  PUT result2;  
RUN;
```

The first use of the function returns the result:

```
Magnificent Car Company
```

Note:

If you had specified `bike` as the *from* string, no changes would have been made, as `bike` would not match `Bike`.

The second use of the function returns the result:

```
Magnificent  Company
```

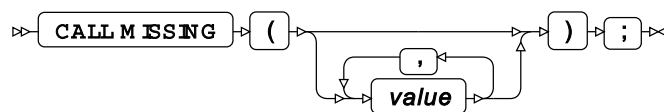
In this case, `Bike` has been replaced with a (' ').

Note:

`TRIMN` is used to ensure any spaces are removed from the null string; this example demonstrates that the null string is replaced by a null in the result of the `TRANWRD` function.

CALL MISSING

Assigns numeric missing values to numeric values, and spaces to string values. By default, numeric missing values are represented by a full stop (`.`). String values are replaced with the same number of spaces as there are characters in the original string.



value

Optional argument

Type: Character or numeric value

The argument to which a missing value is to be applied.

Example

In this example, the function is used to set various values with a corresponding missing value. The result is written to the log.

```
DATA _NULL_;  
  a = 5;  
  b = 6;  
  CALL MISSING(a,b);  
  PUT a;  
  PUT b;  
  c = 'London';  
  d = 'Bike';  
  CALL MISSING(c,d);  
  result1=QUOTE(c);  
  result2=QUOTE(d);  
  PUT result1;  
  PUT result2;  
RUN;
```

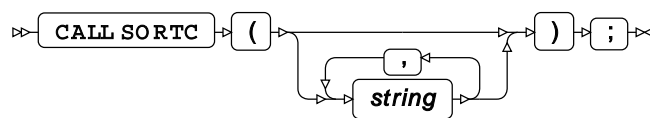
This produces the following output:

```
.  
.  
"      "  
"      "
```

In the first `CALL MISSING`, the numeric values 5 and 6 are both replaced by the numeric missing value . (full stop). In the second `CALL MISSING`, the string values `London` and `Bike` have been replaced by spaces; these strings have the same length as the original string. The `QUOTE` function has been used to wrap the strings in quotation marks, so that the spaces can be seen.

CALL SORTC

Sorts a list of strings.



Sorts a list of strings, presented as arguments, into alphabetic order. The contents of the arguments are changed to match the sort order.

string

Optional argument

Type: Character

An argument containing a string to be sorted.

Basic example

In this example, the function is used to sort a list of strings. The result is written to the log.

```
DATA _NULL_;
  LENGTH c1 c2 c3 $ 11;
  c1 = 'Magnificent';
  c2 = 'London';
  c3 = 'Bikes';
  CALL SORTC(c1,c2,c3);
  PUT c1 " " c2 " " c3;
RUN;
```

This produces the following output:

```
Bikes London Magnificent
```

Example – sorting array of strings

In this example, the function is used to sort the strings in an array. The result is written to the log.

```
DATA _NULL_;
  ARRAY jj(12) $10 ('jack'   'jill'
                   'janet'  'john'
                   'humpty' 'dumpty'
                   'tom'    'jerry'
                   'captain' 'pugwash'
                   'black'  'pig');
  CALL SORTC(of jj(*));
  PUT jj(*);
RUN;
```

This produces the following output:

```
black captain dumpty humpty jack janet jerry jill john pig pugwash tom
```

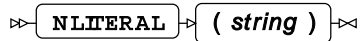
Name literal check and manipulation

Check whether a string is a valid variable name, and convert strings to name literals.

NLITERAL ↗	2100
Returns the string that results from converting a specified string to a name literal.	
NVALID ↗	2100
Returns a value that indicates whether a specified string is a valid variable name.	

NLITERAL

Returns the string that results from converting a specified string to a name literal.



If the specified variable does not have the format of a variable name, the source string is wrapped in double quotation marks and appended with the `N` character. If it does have the format of a valid variable name, the string is returned unchanged.

Return type: Character

string

Type: Character

The string to be converted.

Example

In this example, the function converts two strings to name literals. The result is written to the log.

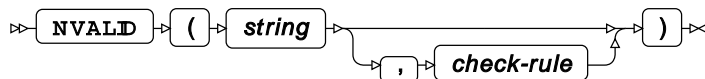
```
DATA _NULL_;
  result1 = NLITERAL('text');
  result2 = NLITERAL('1text');
  PUT result1;
  PUT result2;
RUN;
```

This produces the following output:

```
text
"1text"N
```

NVALID

Returns a value that indicates whether a specified string is a valid variable name.



Return type: Numeric

If the string is a valid name, 1 is returned; 0 is returned otherwise.

string

Type: Character

The string to be analyzed.

check-rule

Optional argument

Specifies how the string should be checked.

"ANY"

Any characters are accepted.

"V7"

A strict interpretation of validity in which strings must meet rules the for variable names introduced at Version 7 of the language of SAS.

Example – valid variable name

In this example, the function checks whether a string is a valid variable name, and returns a value indicating whether it is or not. The result is written to the log.

```
DATA _NULL_;  
  b = 'string1';  
  result = NVALID(b);  
  PUT "Is it a valid variable name? " result;  
RUN;
```

This produces the following output:

```
Is it a valid variable name? 1
```

In this example, `string1` is a valid variable name.

Example – invalid variable name

In this example, the function checks whether a string is a valid variable name, and returns a value indicating whether it is or not. The result is written to the log.

```
DATA _NULL_;  
  b = '1string';  
  result = NVALID(b);  
  PUT "Is it a valid variable name? " result;  
RUN;
```

This produces the following output:

```
Is it a valid variable name? 0
```

In this case, `1string` is an invalid variable name.

Example – using ANY to allow invalid variable names

In this example, the function checks whether a string is a valid variable name, and returns a value indicating whether it is or not. The result is written to the log.

```
DATA _NULL_;
  b = '1string';
  result = NVALID(b, 'ANY');
  PUT "Is it a valid variable name? " result;
RUN;
```

This produces the following output:

```
Is it a valid variable name? 1
```


Although `1string` is an invalid variable name, the `ANY` argument allows any characters to be accepted as valid.

Remove spaces from a source string

Removes spaces from a specified string.

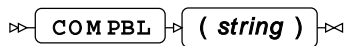
You can compress multiple spaces to single spaces, remove leading and trailing spaces, and so on.

COMPBL 🔗	2103
Returns the string that results from replacing each instance of multiple consecutive spaces with a single space. Tabs are unaffected.	
LEFT 🔗	2103
Return the string that results from removing leading (left-hand) spaces from a specified string.	
KLEFT 🔗	2104
Returns a specified string of DBCS characters with any leading (left-hand) spaces removed.	
KRIGHT 🔗	2105
Returns a specified string of DBCS characters with any trailing (right-hand) spaces removed.	
KTRIM 🔗	2106
Returns the string that results from trimming trailing spaces from a string consisting of DBCS characters.	
RIGHT 🔗	2107
Returns a string with any trailing (right-hand) spaces removed and prepended to the left-hand side.	
STRIP 🔗	2108
Returns a specified string and removes any leading and trailing spaces it might have. No other spaces are stripped.	
TRIM 🔗	2109
Returns a string that contains the source string trimmed of trailing spaces. No other spaces are stripped.	

TRIMN 	2109
Returns a string that contains the source string trimmed of trailing spaces. No other spaces are stripped. The source string can be a null string.	

COMPBL

Returns the string that results from replacing each instance of multiple consecutive spaces with a single space. Tabs are unaffected.



Return type: Character

string

Type: Character

The string to be examined.

Example

In this example, the function is used to replace multiple consecutive spaces with single spaces. There is a tab between 2 and 50 in the string. This produces the following output:

```

DATA _NULL_;
  result = COMPBL("Steve Bike      Company      0      1 2 50");
  PUT "The string returned is: " result;
RUN;
  
```

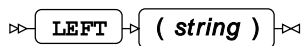
This produces the following output:

```

The string returned is: Steve Bike Company 0 1 2 50
  
```

LEFT

Return the string that results from removing leading (left-hand) spaces from a specified string.



The spaces will be appended to the right-hand side of the source string. Whether the appended spaces are retained with the string depends on subsequent functions and actions; see section below for examples.

Return type: Character

string**Type:** Character

A string from which leading spaces will be removed.

Example

In this example, the function is used to remove the leading spaces from a source string. The result is written to the log.

The `QUOTE` function is used to apply quotes to the returned string, enabling you to see that spaces have been moved to the trailing (right-hand) position. The `PUT` statements add the word `Metal`; the second `PUT` shows how, in this instance, the appended spaces are lost. The third `PUT` statement shows the result of concatenating the `LEFT` result with the string `Metal`; again, this demonstrates that the leading spaces have been removed and appended as trailing spaces.

```
DATA _NULL_;  
  text='    Bike Pedals';  
  result = quote(left(text));  
  result2 = left(text);  
  result3 = cat(left(text), 'Metal');  
  PUT result 'Metal';  
  PUT result2 'Metal';  
  PUT result3;  
RUN;
```

```
"Bike Pedals    " Metal  
Bike Pedals Metal  
Bike Pedals    Metal
```

KLEFT

Returns a specified string of DBCS characters with any leading (left-hand) spaces removed.

➤ **KLEFT** ➤ (➤ ***string*** ➤) ➤

Removes any leading spaces at the left-hand of a source string consisting of characters from a double-byte character set (DBCS). The spaces will be appended to the right of the source; whether the appended spaces are returned with the string depends on subsequent functions and actions; see the section below for examples.

Return type: Character***string*****Type:** Character

A string from which leading spaces will be removed.

Example

In this example, the function is used to remove the leading spaces from a source string. The result is written to the log. The `QUOTE` function is used to apply quotation marks to the returned string, enabling you to see that spaces have been moved to the trailing (right-hand) position. The `PUT` statements add the string `>熨斗目花色>`; the second `PUT` shows how in this instance the appended spaces are lost. The third `PUT` statement shows the result of concatenating the `LEFT` result with the string `>熨斗目花色>`; again, this demonstrates that the leading spaces have been removed and appended as trailing spaces.

```
DATA _NULL_;
  text='      熨斗目花色';
  result = QUOTE(KLEFT(text));
  result2 = KLEFT(text);
  result3 = CAT(KLEFT(text), '熨斗目花色');
  PUT result '熨斗目花色';
  PUT result2 '熨斗目花色';
  PUT result3;
RUN;
```

This produces the following output:

```
"熨斗目花色      " 熨斗目花色
熨斗目花色 熨斗目花色
熨斗目花色 熨斗目花色
```

In the first result, quoting the returned string (using the `QUOTE` function) has retained the appended spaces. In the second result, concatenating the strings using the `PUT` function has removed the appended spaces. In the third result, concatenating strings using the `CAT` function has retained the appended spaces.

KRIGHT

Returns a specified string of DBCS characters with any trailing (right-hand) spaces removed.

```
⇒ KRIGHT ⇒ ( ⇒ string ⇒ ) ⇒
```

Removes any trailing spaces at the right-hand of a source string consisting of characters from a double-byte character set (DBCS).

The spaces will be prepended at the left-hand side of the source, becoming leading spaces. Whether the prepended spaces are retained depends on subsequent functions and actions; see the section below for examples.

Return type: Character

string

Type: Character

The string from which trailing spaces are to be removed.

Example

In this example, the function is used to remove the trailing spaces from a source string. The result is written to the log. The second `PUT` statement adds the word `Metal`; this second example shows how the appended spaces are lost. The third `PUT` statement shows the result of concatenating the `RIGHT` result with the string `Metal`; this demonstrates that the trailing spaces have been removed and appended as leading spaces.

```
DATA _NULL_;
  text='  猩々緋      ';
  result = KRIGHT(text);
  result1 = QUOTE(KRIGHT(text));
  result2 = CAT('熨斗目花色', KRIGHT(text));
  PUT result;
  PUT result1;
  PUT '熨斗目花色      ' result;
  PUT result2;
RUN;
```

This produces the following output:

```
猩々緋
"      猩々緋"
熨斗目花色 猩々緋
熨斗目花色 猩々緋
```

In the first result, the `PUT` has removed the prepended spaces. In the second result, quoting the returned string (using the `QUOTE` function) has retained the prepended spaces. In the third and fourth results, the concatenation of the strings has also retained the prepended spaces.

KTRIM

Returns the string that results from trimming trailing spaces from a string consisting of DBCS characters.

⇒ **KTRIM** ⇒ (⇒ *string* ⇒) ⇒

Removes trailing spaces from a source string consisting of characters from a double-byte character set (DBCS), and returns the modified string. No other spaces are stripped. A trailing horizontal tab is recognised as a space, and so will also be removed.

Return type: Character

string

Type: Character

The string to be stripped of trailing spaces.

Example

In this example, the function is used to strip spaces from a string. The trailing spaces consist of a tab and two spaces. The result is written to the log.

```
DATA _NULL_;  
  result= QUOTE(KTRIM("  青碧 熨斗目花色 、 猩々緋 青柳鼠  "));  
  PUT result;  
RUN;
```

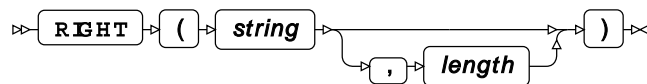
This produces the following output:

```
"  青碧 熨斗目花色 、 猩々緋 青柳鼠"
```

KTRIM has removed all trailing spaces, including tabs, while retaining leading spaces. QUOTE has wrapped the string in double quotation marks, enabling the retained spaces to be seen.

RIGHT

Returns a string with any trailing (right-hand) spaces removed and prepended to the left-hand side.



Whether the prepended spaces are retained, and displayed or printed, depends on subsequent functions and actions; see the section below for examples.

Return type: Character

string

Type: Character

The string from which trailing spaces are to be removed.

length

Optional argument

Type: Numeric

Not yet available.

Example

In this example, the function is used to remove the trailing spaces from a source string. The result is written to the log. The second `PUT` statement adds the word `Metal`; this second example shows how the prepended spaces are lost. The third `PUT` statement shows the result of concatenating the `RIGHT` result with the string `Metal`; this demonstrates that the trailing spaces have been removed and prepended as leading spaces.

```
DATA _NULL_;
  text = 'Bike Pedals   ';
  result1 = RIGHT(text);
  result2 = CAT('Metal', right(text));
  PUT result1;
  PUT 'Metal ' result;
  PUT result2;
RUN;
```

This produces the following output:

```
Bike Pedals
Metal Bike Pedals
Metal   Bike Pedals
```

STRIP

Returns a specified string and removes any leading and trailing spaces it might have. No other spaces are stripped.

➤ **STRIP** ➤ (*string*) ➤

Return type: Character

string

Type: Character

The string to be stripped of leading and trailing spaces.

Example

In this example, the function is used to strip leading and trailing spaces from a string. The result is written to the log.

```
DATA _NULL_;
  result = STRIP('   London Bike Company ');
  PUT 'The space-stripped string is: ' result;
RUN;
```

This produces the following output:

```
The space-stripped string is: London Bike Company
```

TRIM

Returns a string that contains the source string trimmed of trailing spaces. No other spaces are stripped.

➤ **TRIM** ➤ (*string*) ➤

Return type: Character

string

Type: Character

The string to be stripped of trailing spaces.

Example

In this example, the function is used to strip spaces from a string. The `QUOTE` function has also been used to wrap the result in quotations marks so remaining leading spaces can be seen. The result is written to the log.

```
DATA _NULL_;  
  result= QUOTE(TRIM("  Magnificent Bike Company A 1 2 50   London   "));  
  PUT result;  
RUN;
```

This produces the following output:

```
"  Magnificent Bike Company A 1 2 50   London"
```

`TRIM` has removed all trailing spaces, while retaining leading spaces.

Note:

A horizontal tab is not recognised as a space, and so will not be trimmed.

TRIMN

Returns a string that contains the source string trimmed of trailing spaces. No other spaces are stripped. The source string can be a null string.

➤ **TRIMN** ➤ (*string*) ➤

If the string is a null string, this function does not return a single space.

Note:

A horizontal tab is not recognised as a space, so is not be trimmed.

Return type: Character

string

Type: Character

The string to be stripped of trailing spaces.

Example

In this example, the function is used to strip spaces from a string. The `QUOTE` function has also been used to wrap the result in quotation marks to make resulting spaces visible. The result is written to the log.

```
DATA _NULL_;
  a='';
  result1 = QUOTE(TRIMN('    Bike    '));
  result2 = QUOTE(TRIM(a));
  result3 = QUOTE(TRIMN(a));

  PUT result1;
  PUT result2;
  PUT result3;
RUN;
```

This produces the following output:

```
"    Bike"
" "
" "
```

In the first result, the trailing spaces have been removed, while leading spaces have been retained

The second and third results compare the use of `TRIM` and `TRIMN` on a null. For `TRIM`, a space is returned. For `TRIMN`, a null is returned.

System command function and `CALL` routine

Execute system commands and run executable files.

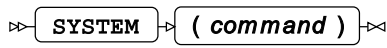
You can execute operating system commands such as `DIR` or `COPY` on Windows, or `cd` or `rmdir` on Linux. You can also run program executables if paths are specified or set beforehand.

SYSTEM [↗](#).....2111
Executes an operating system command or a program executable and returns a value.

CALL SYSTEM ↗	2112
Executes an operating system command or a program executable.	

SYSTEM

Executes an operating system command or a program executable and returns a value.



If the system option `XSsync` is set, no other `DATA` step instructions are executed until the process finishes, which might involve the user closing any windows that have been opened by the system process.

`XSsync` is set by default when WPS is installed. If you set the option to `NOXSsync`, the `DATA` step finishes execution whether the system process has completed or not. See `XSsync` [↗](#) (page 90) for more information.

Return type: Numeric

command

Type: Character

The system command or executable name.

Basic example

In this example, the Windows command prompt window is displayed. The result is written to the log. The program does not execute the next statement (the `PUT`), until the command prompt window is closed.

```

DATA _NULL_;

  syscmd=SYSTEM( "CMD" ) ;
  PUT syscmd;

RUN;
  
```

This produces the following output:

```

-1073741510
  
```

Example – running program executable

In this example, the Notepad++ program is started. The result is written to the log. Because the system option `NOXSYNC` has been specified, the program does not pause after each use of the function.

```
OPTION NOXSYNC;
DATA _NULL_;

  x = SYSTEM('set PATH=%PATH%;"C:\Program Files (x86)\Notepad++"');
  x = SYSTEM('Notepad++.exe');
  put x;

RUN;
```

This produces the following output:

```
0
```

CALL SYSTEM

Executes an operating system command or a program executable.

➤ **CALL SYSTEM** ➤ (*command*) ➤ ; ➤

If the system option `XSYNC` is set, no other `DATA` step instructions are executed until the process finishes, which might involve the user closing any windows that have been opened by the system process.

`XSYNC` is set by default when WPS is installed. If you set the option to `NOXSYNC`, the `DATA` step finishes execution whether the system process has completed or not. See [XSUNC](#) (page 90) for more information.

command

Type: Character

The system command.

Basic example

In this example, a Windows `DIR` command is executed. The content of the specified folder is displayed in a Windows CMD window. The `DATA` step does not execute the next statement (or reach the end of the step), until the operating system process is closed or terminates.

```
DATA _NULL_;

  CALL SYSTEM('DIR C:\');

RUN;
```

Example – running program executable

In this example, the Notepad++ program is started. The result is written to the log. Because the system option NOXSYNC has been specified, the program does not pause after each use of the routine.

```
OPTION NOXSYNC;
DATA _NULL_;

CALL SYSTEM('set PATH=%PATH%;"C:\Program Files (x86)\Notepad++"');
CALL SYSTEM('Notepad++.exe');

RUN;
```

System information functions

Return information about the operating system and WPS.

ENVLEN ↗	2113
Returns the length of the value of an environment variable.	
GETOPTION ↗	2114
Returns the current settings of a WPS system option.	
SYSGET ↗	2115
Returns the value of the specified environment variable.	
SYSPARM ↗	2116
Returns the value of a system environment parameter.	
SYSPROCESSID ↗	2118
Returns the current system process identifier.	
SYSPROCESSNAME ↗	2118
Returns the name of the current system process.	
SYSPROD ↗	2119
Returns an indicator to determine whether a product license is valid.	

ENVLEN

Returns the length of the value of an environment variable.



Return type: Numeric

The variable length.

environment-variable**Type:** Character

The variable.

Example

In this example, the function is used to find the length of five environment variables. The result is written to the log.

```
DATA _NULL_;

var_path=ENVLEN('PATH');
var_temp=ENVLEN('TEMP');
var_tmp=ENVLEN('TMP');
var_username=ENVLEN('USERNAME');
var_psmppath=ENVLEN('PSModulePath');

PUT "The length for the environment variable 'PATH' is:           " var_path;
PUT "The length for the environment variable 'TEMP' is:          " var_temp;
PUT "The length for the environment variable 'TMP' is:           " var_tmp;
PUT "The length for the environment variable 'USERNAME' is:      " var_username;
PUT "The length for the environment variable 'PSModulePath' is:  " var_psmppath;

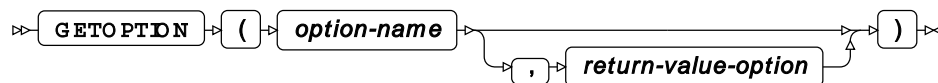
RUN;
```

This produces the following output:

```
The length for the environment variable 'PATH' is:           937
The length for the environment variable 'TEMP' is:          38
The length for the environment variable 'TMP' is:           38
The length for the environment variable 'USERNAME' is:      10
The length for the environment variable 'PSModulePath' is:  93
```

GETOPTION

Returns the current settings of a WPS system option.

**Return type:** Character**option-name****Type:** Character

The name of the system option.

return-value-option

Optional argument

Type: Character

Enter `HOWSET`, to obtain the location of the source information, or enter `HOWRESTRICTED`, to determine whether any restriction applies to the WPS system option. By default, only *option-name* is returned.

Example 1 – using the default return value option

In this example, the current WPS system option is returned. The result is written to the log.

```
DATA _NULL_;  
  col1=GETOPTION("YEARCUTOFF");  
  col2=GETOPTION("ENGINE");  
  col3=GETOPTION("EMAILSYS");  
  col4=GETOPTION("EMAILPORT");  
  col5=GETOPTION("ENCODING");  
  PUT col1 col2 col3 col4 col5;  
RUN;
```

This produces the following output:

```
1926 WPD MAPI 25 UTF-8
```

Example 2 – specifying a value for the return value option

In this example, the `return-value-option` is used to either find the source location of a WPS system option, or to determine whether any restrictions apply to the WPS system option. The result is written to the log.

```
DATA _NULL_;  
  col1=GETOPTION("YEARCUTOFF", "HOWSET");  
  col2=GETOPTION("ENGINE", "HOWSET");  
  col3=GETOPTION("EMAILSYS", "HOWSET");  
  col4=GETOPTION("EMAILPORT", "HOWRESTRICTED");  
  col5=GETOPTION("ENCODING", "HOWRESTRICTED");  
  PUT col1 col2 col3 col4 col5;  
RUN;
```

This produces the following output:

```
Config file Config file Built-in default Unrestricted Unrestricted
```

SYSGET

Returns the value of the specified environment variable.

➤ **SYSGET** ➤ (*environment-variable*) ➤

Return type: Character

The value of the environment variable.

environment-variable

Type: Character

The name of the environment variable.

Example

In this example, the function is used to return the value for four environment variables. The result is written to the log.

```
DATA _NULL_;

var_win=SYSGET("WINDIR");

var_os=SYSGET("OS");

var_proc_arch=SYSGET("PROCESSOR_ARCHITECTURE");

var_user_name=SYSGET("USERNAME");

PUT "The value for WINDIR is:           " var_win;
PUT "           ";
PUT "The value for OS is:             " var_os;
PUT "           ";
PUT "The value for PROCESSOR_ARCHITECTURE is: " var_proc_arch;
PUT "           ";
PUT "The value for USERNAME is:       " var_user_name;
PUT "           ";

RUN;
```

This produces the following output:

```
The value for WINDIR is:           C:\WINDOWS
The value for OS is:             Windows_NT
The value for PROCESSOR_ARCHITECTURE is: AMD64
The value for USERNAME is:       tom.phillips
```

SYSPARM

Returns the value of a system environment parameter.

➤ **SYSPARM** ➤ **()** ➤

Return type: Character

The value of the environment parameter.

When set, the system environment parameter is unique and the information can be shared in many programs running during the same session. It can be reset by running an update program with a new value, or restarting WPS (in which case the information is removed).

Example – finding a value of the system environment parameter set by `OPTIONS`

In this example, the system environment parameter is set up using the `OPTIONS` statement and then tested in the `DATA` step. The result is written to the log.

```
OPTIONS SYSPARM="hello world";

DATA _NULL_;
  FORMAT p $32.;

  p = SYSPARM();

  PUT p;

RUN;
```

This produces the following output:

```
hello world
```

Example – finding a value of the system environment parameter set by a macro

In this example, an environment parameter is set up by a macro and then tested in the `DATA` step. The result is written to the log.

```
%Global environmentParameter;
%Let SYSPARM=Port=3000;

DATA _NULL_;

  param = SYSPARM();
  PUT param;

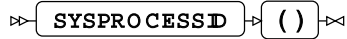
RUN;
```

This produces the following output:

```
Port=3000
```

SYSPROCESSID

Returns the current system process identifier.



Return type: Character

A process identifier.

Example

In this example, the current system process identifier is returned. The result is written to the log.

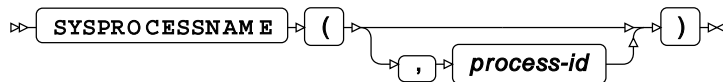
```
DATA _NULL_;  
  sysid = SYSPROCESSID();  
  PUT "System Process Id = " sysid;  
RUN;
```

This produces the following output:

```
System Process Id = 41DB229294FE56040000000000000000
```

SYSPROCESSNAME

Returns the name of the current system process.



Return type: Character

A product name.

process-id

Optional argument

Type: Character

A process identifier.

Example

In this example, both the current system process identification and name are returned. The result is written to the log.

```
DATA _NULL_;  
  sysid = SYSPROCESSID();  
  sysnameid = SYSPROCESSNAME();  
  PUT "System Process Id = " sysid;  
  PUT "System Process Name = " sysnameid;  
RUN;
```

This produces the following output:

```
System Process Id = 41DB2C313F37DF3B0000000000000000  
System Process Name = Wpslinks Server
```

SYSPROD

Returns an indicator to determine whether a product license is valid.

➤ **SYSPROD** ➤ (*product-name*) ➤

Return type: Numeric

1 if valid, or -1 for not available.

product-name

Type: Character

The product name.

Example

In this example, products are checked to determine whether a license is available. The result is written to the log.

```
DATA _NULL_;  
  
  a = SYSPROD("WPS");  
  b = SYSPROD("WPSWEB");  
  c = SYSPROD("ORACLE");  
  d = SYSPROD("COBOL");  
  e = SYSPROD("C++");  
  f = SYSPROD("GRAPH");  
  
  if a = 1 THEN PUT "A license is available for WPS ";  
  ELSE PUT "A license is not available for WPS ";  
  
  if b = 1 THEN PUT "A license is available for WPSWEB ";  
  ELSE PUT "A license is not available for WPSWEB ";
```

```
if c = 1 THEN PUT "A license is available for ORACLE ";
ELSE PUT "A license is not available for ORACLE ";

if d = 1 THEN PUT "A license is available for COBOL ";
ELSE PUT "A license is not available for COBOL ";

if e = 1 THEN PUT "A license is available for C++ ";
ELSE PUT "A license is not available for C++ ";

if f = 1 THEN PUT "A license is available for GRAPH ";
ELSE PUT "A license is not available for GRAPH ";
RUN;
```

This produces the following output:

```
A license is available for WPS
A license is not available for WPSWEB
A license is available for ORACLE
A license is not available for COBOL
A license is not available for C++
A license is available for GRAPH
```

Truncation and rounding functions

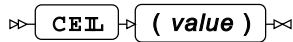
Define how numbers will be truncated or rounded.

CEIL ↗	2121
Returns the next higher integer for a decimal number, unless the fraction is within 1E-12 of the next lower integer, in which case the next lower integer is returned.	
CEILZ ↗	2124
Returns the next higher integer for a decimal number.	
FLOOR ↗	2125
Returns the integer part of a decimal number, unless the fractional part is within 1E-12 of the next higher integer, in which case that integer is returned.	
FLOORZ ↗	2127
Returns the integer part of a decimal number.	
FUZZ ↗	2128
Returns the specified decimal number, unless the fractional part is within 1E-12 (the epsilon) of the next higher or next lower integer, in which case that integer is returned.	
INT ↗	2131
Returns the integer part of a specified decimal number, unless the fractional part is within 1E-12 (the epsilon) of the next higher (if positive) or next lower (if negative) integer, in which case that integer is returned.	
INTZ ↗	2135
Returns the integer part of a specified decimal number.	

ROUND ↗	2136
Returns the rounded value of a specified number. If the value is within a very small fraction (an epsilon) of the midpoint of the numbers to which the specified number could be rounded, the result is treated as being at the midpoint and rounded up.	
ROUNDE ↗	2139
Returns the rounded value of a specified number. If the value falls exactly halfway, or is within a very small fraction (the epsilon) of halfway between numbers to which the specified number could be rounded, the nearest even multiple of the specified unit is returned.	
ROUNDZ ↗	2142
Returns the rounded value of a specified number. If the value falls exactly halfway between numbers to which the specified number could be rounded, the nearest even multiple of the specified unit is returned.	
TRUNC ↗	2144
Returns the number that results from truncating the floating point representation of a specified number to a defined number of bytes.	

CEIL

Returns the next higher integer for a decimal number, unless the fraction is within 1E-12 of the next lower integer, in which case the next lower integer is returned.



Return type: Numeric

An integer.

value

Type: Numeric

The decimal number to be rounded to the next higher integer.

CEIL can be visualised as acting on a number line:



If a number falls within the grey area (the epsilon), the integer n is returned. If a number falls elsewhere on the number line, the integer $n+1$ is returned. For example, if *value* is:

- 1.09, then 2 is returned, as this is the next higher integer
- 1.00000000000009, then 1 is returned, as the fractional part falls below the 1E-12 boundary, and 1 is the next lower integer
- -1.99, then -1 is returned, as this is the next higher integer
- -1.99999999999999, then -2 is returned, as the fractional part falls below the 1E-12 boundary, and -2 is the next lower integer

If you want to return the next higher integer whatever the magnitude of the fractional part of the number, see `CEILZ` [↗](#) (page 2124).

Basic example

In this example, various numbers are truncated; the function return either the next higher integer, depending on whether the fractional part falls within the epsilon for the function. The result is written to the log.

```
RUN;

DATA _NULL_;

    tot = 2 + 1e-6;
    cl = CEIL(tot);
    PUT "Because the number " tot "is above the epsilon, returns: " cl;

    tot = 2 + 1E-14;
    cl = CEIL(tot);
    PUT "Because the number " tot "16.14 " is within the epsilon, returns: " cl;

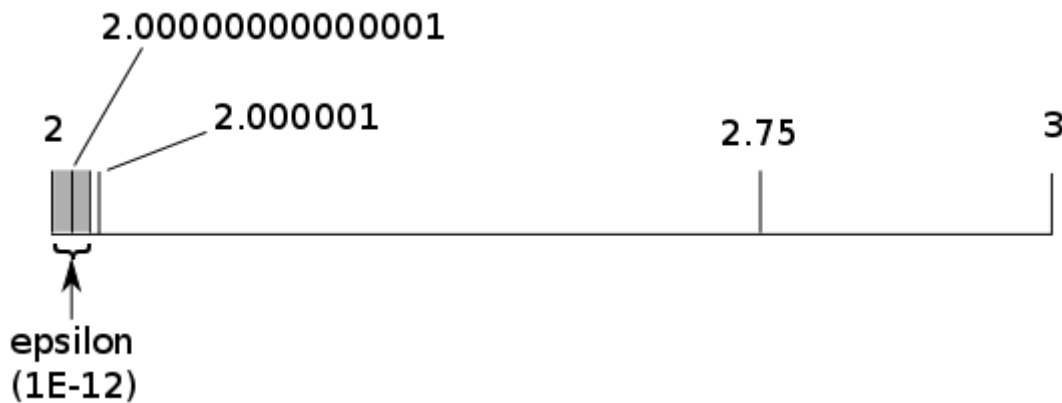
    tot = 2 + 0.75;
    cl = CEIL(tot);
    PUT "Because the number " tot "is above the epsilon, returns: " cl;

RUN;
```


This produces the following output:

```
Because the number 2.000001 is above the epsilon, returns: 3
Because the number 2.0000000000000010 is within the epsilon, returns: 2
Because the number 2.75 is above the epsilon, returns: 3
```

These results can be visualised on the following number line:



2.000000000000001 falls within the epsilon, so the lower integer is returned. The other numbers are above the epsilon, so the higher integer is returned.

Example – positive and negative numbers

In this example, various numbers are truncated; the function returns either the next higher or next lower integer, depending on whether the fractional part falls within the epsilon for the function. The result is written to the log.

```
DATA _NULL_;

tot = 2 + 1e-12;
cl = CEIL(tot);
PUT "Because the number " tot "is on the upper boundary, returns: " cl;

tot = -2 + 1E-14;
cl = CEIL(tot);
PUT "Because the number " tot " is within the epsilon, returns: " cl;

tot = -2 + 0.75;
cl = CEIL(tot);
PUT "Because the number " tot 4.2 " is above the epsilon, returns: " cl;

RUN;
```

This produces the following output:

```
Because the number 2 is on the upper boundary, returns: 3
Because the number -1.999999999999999 is within the epsilon, returns: -2
Because the number -1.3 is above the epsilon, returns: -1
```

Because the first number falls exactly on the upper boundary of the epsilon, the number returned is the next higher integer. In the second use of the function, the number is within the epsilon of the next lower integer, so that integer is returned. In the third use of the function, the number is above the epsilon, so the next higher integer is returned.

CEILZ

Returns the next higher integer for a decimal number.

➤ **CEILZ** ➤ **(value)** ➤

Return type: Numeric

An integer.

value

Type: Numeric

The decimal number to be rounded to the next higher integer.

If you want to return the next lower integer when the fractional part of the number is within a very close range (less than 1E-12) of that integer, see [CEIL](#) (page 2121).

Example

In this example, numbers are rounded to the next higher integer. The result is written to the log.

```
DATA _NULL_;

  nhi = CEILZ(2.0000000000000001);
  PUT "The next higher integer is: " nhi;

  nhi = CEILZ(1.5);
  PUT "The next higher integer is: " nhi;

  nhi = CEILZ(-1.5);
  PUT "The next higher integer is: " nhi;

RUN;
```

This produces the following output:

```
The next higher integer is: 2
The next higher integer is: 2
The next higher integer is: -1
```

FLOOR

Returns the integer part of a decimal number, unless the fractional part is within 1E-12 of the next higher integer, in which case that integer is returned.

➤ `FLOOR` ➤ `(value)` ➤

Return type: Numeric

An integer.

value

Type: Numeric

The decimal number to be rounded to the integer part.

`FLOOR` can be visualised as acting on a number line:



If a number falls within the grey area (the epsilon), the integer n is returned. If a number falls elsewhere on the number line, the integer $n-1$ is returned. For example, if *value* is:

- 1.75, then 1 is returned, as this is the next lower integer
- 1.99999999999999, then 2 is returned, as the fractional part falls above the 1E-12 boundary, and 2 is the next higher integer
- -1.01, then -2 is returned, as this is the next lower integer
- -1.00000000000001, then -1 is returned, as the fractional part falls above the 1E-12 boundary, and -1 is the next higher integer

If you want to return the next lower integer whatever the magnitude of the fractional part of the number, see `FLOORZ` [↗](#) (page 2127).

Basic example

In this example, various numbers are truncated; the function return either the next higher or next lower integer, depending on whether the fractional part falls within the epsilon for the function. The result is written to the log.

```
DATA _NULL_;

tot = 2 + 0.999999;
fl = FLOOR(tot);
PUT "Because the number " tot 9.6 " is below the epsilon, returns: " fl;

tot = 2 + 0.999999999999999;
fl = FLOOR(tot);
PUT "Because the number " tot 15.13 " is within the epsilon, returns: " fl;

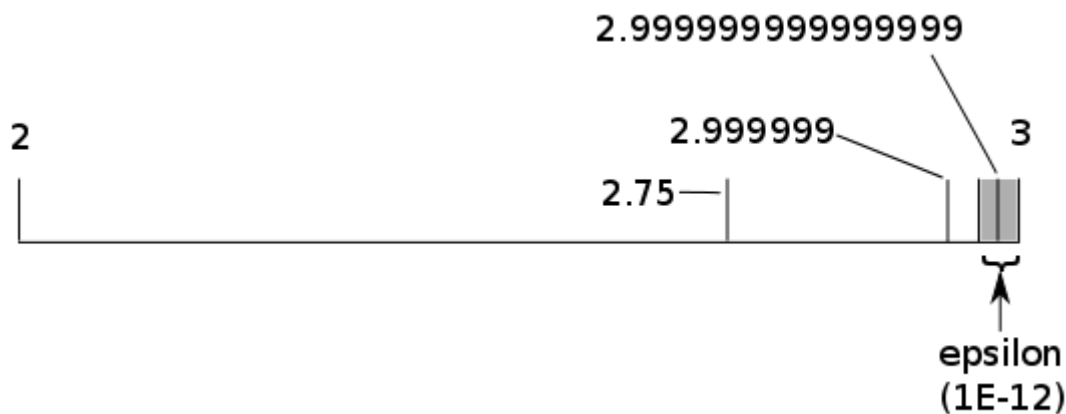
tot = 2 + 0.75;
fl = FLOOR(tot);
PUT "Because the number " tot "is below the epsilon, returns: " fl;

RUN;
```

This produces the following output:

```
Because the number 2.999999 is below the epsilon, returns: 2
Because the number 2.999999999999999 is within the epsilon, returns: 3
Because the number 2.75 is below the epsilon, returns: 2
```

These results can be visualised on the following number line:



2.999999999999999 falls within the epsilon, so the higher integer is returned. The other numbers are below the epsilon, so the lower integer is returned.

Example – positive and negative numbers

In this example, various numbers are truncated; the function return either the next higher or next lower integer, depending on whether the fractional part falls within the epsilon for the function. The result is written to the log.

```
DATA _NULL_;

tot = -2-1e-12;
fl = FLOOR(tot);
PUT "Because the number " tot 16.12 " is on the upper boundary, returns: " fl;

tot = -2 -1e-13;
fl = FLOOR(tot);
PUT "Because the number " tot 16.13 " is within the epsilon, returns: " fl;

tot = -2-0.75;
fl = FLOOR(tot);
PUT "Because the number " tot 4.2 " is above the epsilon, returns: " fl;

RUN;
```

This produces the following output:

```
Because the number -2.0000000000001 is on the upper boundary, returns: -3
Because the number -2.0000000000001 is within the epsilon, returns: -2
Because the number -2.8 is above the epsilon, returns: -3
```

Because the first number falls exactly on the upper boundary of the epsilon, the number returned is the next lower integer. In the second use of the function, the number is within the epsilon of the previous integer (-2), so that integer is returned. In the third use of the function, the number is below the epsilon, so the next lower integer (-3) is returned.

FLOORZ

Returns the integer part of a decimal number.

➤ **FLOORZ** ➤ (*value*) ➤

Return type: Numeric

An integer.

value

Type: Numeric

The decimal number to be rounded to the integer part.

If you want to return the next higher integer when the fractional part of the number is within a very close range (less than 1E-12) of that integer, use [FLOOR](#) (page 2125).

Example

In this example, various numbers are rounded. The result is written to the log.

```
DATA _NULL_;  
  
  rc = FLOORZ (2.5);  
  PUT rc = ;  
  
  rc = FLOORZ (2.9999999999991);  
  PUT rc = ;  
  
  rc = FLOORZ (-2.5);  
  PUT rc = ;  
  
RUN;
```

This produces the following output:

```
rc=2  
rc=2  
rc=-3
```

FUZZ

Returns the specified decimal number, unless the fractional part is within 1E-12 (the epsilon) of the next higher or next lower integer, in which case that integer is returned.

➤ **FUZZ** ➤ (*value*) ➤

Return type: Numeric

An integer, or *value*.

value

Type: Numeric

A number to be rounded.

FUZZ can be visualised as acting on a number line:



If the specified number falls within the epsilon, the corresponding lower or higher integer is returned. If the specified number falls elsewhere on the number line, that number is returned. For example, if *value* is:

- 1.75, then 1.75 is returned
- 1.999999999999, then 2 is returned, as the fractional part falls within the epsilon, and 2 is the next higher integer
- -1.01, then -1.01 is returned
- -1.999999999999, then -2 is returned, as the fractional part falls within the epsilon, and -2 is the next lower integer

Basic example

In this example, the next higher or lower integer is returned for numbers where the fractional part falls within the epsilon of the numbers, otherwise the specified numbers are returned. The result is written to the log.

```
DATA _NULL_;

tot = 2 + 0.75;
fz = FUZZ(tot);
PUT "Because the number " tot 4.2" is not within either epsilon, returns: " fz;

tot = 2 + 0.00000001;
fz = FUZZ(tot);
PUT "Because the number " tot 9.7" is not within either epsilon, returns: " fz;

tot = 2 + 0.99999999999999;
fz = FUZZ(tot);
PUT "Because the number " tot 15.13 " is within the upper epsilon, returns: " fz;

tot = 2 + 0.999999;
fz = FUZZ(tot);
PUT "Because the number " tot 8.6 " is not within either epsilon, returns: " fz;

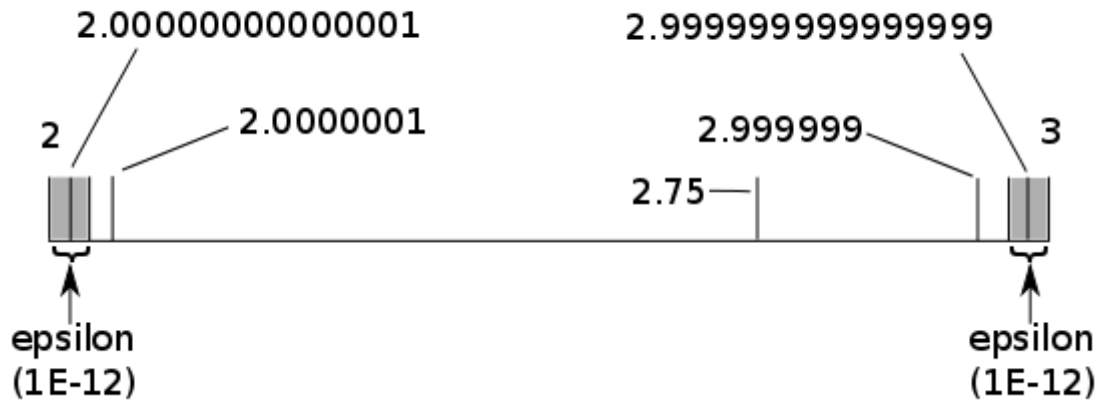
tot = 2 + 0.0000000000000001;
fz = FUZZ(tot);
PUT "Because the number " tot 15.13 " is within the lower epsilon, returns: " fz;

RUN;
```

This produces the following output:

```
Because the number 2.75 is not within either epsilon, returns: 2.75
Because the number 2.0000001 is not within either epsilon, returns: 2.0000001
Because the number 2.9999999999999 is within the upper epsilon, returns: 3
Because the number 2.999999 is not within either epsilon, returns: 2.999999
Because the number 2.00000000000001 is within the lower epsilon, returns: 2
```

These results can be visualised on the following number line:



2.999999999999999 falls within the upper epsilon, so the higher integer (3) is returned.
2.00000000000001 falls within the lower epsilon, so the lower integer (2) is returned. The other numbers not within either epsilon, so the specified number is returned.

Example – positive and negative numbers

In this example, various numbers are truncated; the function returns either the next lower integer, depending on whether the fractional part falls within the epsilon for the function. The result is written to the log.

```
DATA _NULL_;

    tot = -2-1e-12;
    fz = FUZZ(tot);
    PUT "Because " tot 15.12 " is on boundary of upper epsilon, returns number: " fz
    15.12;

    tot = -2-1e-13;
    fz = FUZZ(tot);
    PUT "Because " tot 16.13 " is within epsilon, returns higher integer: " fz;

    tot = -2-0.999999999999999;
    fz = FUZZ(tot);
    PUT "Because " tot 17.14 " is within epsilon, returns lower integer: " fz;

    tot = -2-0.75;
    fz = FUZZ(tot);
    PUT "Because " tot 5.2 " is outside an epsilon, returns number: " fz;

RUN;
```

This produces the following output:

```
Because -2.000000000000001 is on boundary of upper epsilon, returns number:
-2.000000000000001
Because -2.000000000000001 is within the epsilon, returns higher integer: -2
Because -2.999999999999990 is within the epsilon, returns lower integer: -3
Because -2.75 is outside an epsilon, returns number: -2.75
```

Because the first number falls exactly on the boundary of the upper epsilon, the specified number is returned. In the second use of the function, the number is within the epsilon of the higher integer (-2), so that integer is returned. In the third use of the function, within the epsilon of the lower integer (-3) so that integer is returned.

INT

Returns the integer part of a specified decimal number, unless the fractional part is within 1E-12 (the epsilon) of the next higher (if positive) or next lower (if negative) integer, in which case that integer is returned.

➤ **INT** ➤ (value) ➤

Return type: Numeric

An integer.

value**Type:** Numeric

The number to be rounded.

`INT` can be visualised as acting on a number line. For positive numbers:

For negative numbers:



If the specified number falls within a grey area, the corresponding lower or higher integer n is returned. If the specified number falls elsewhere on the number line, the integer part of that number is returned. For example, if *value* is:

- 1.75, then 1 is returned
- 1.9999999999999999, then 2 is returned, as the fractional part falls within the epsilon for positive numbers, and 2 is the next higher integer
- -1.01, then -1 is returned
- -1.9999999999999999, then -2 is returned, as the fractional part falls within the epsilon for negative numbers, and -2 is the next lower integer

If you want to return the integer of the specified value whatever the magnitude of the fractional part of the number, use `INTZ` [↗](#) (page 2135).

Example – with positive numbers

In this example, the integer part of decimal numbers is returned, unless the fractional part falls within the epsilon of the next higher integer, in which case that integer is returned instead. The result is written to the log.

```
DATA _NULL_;

  tot = 2 + 0.75;
  in = INT(tot);
  PUT "Because " tot "is outside of epsilon, returns integer part of number: " in;

  tot = 2 + 1e-12;
  in = INT(tot);
  PUT "Because " tot " is on boundary of upper epsilon, returns integer part of number: " in;

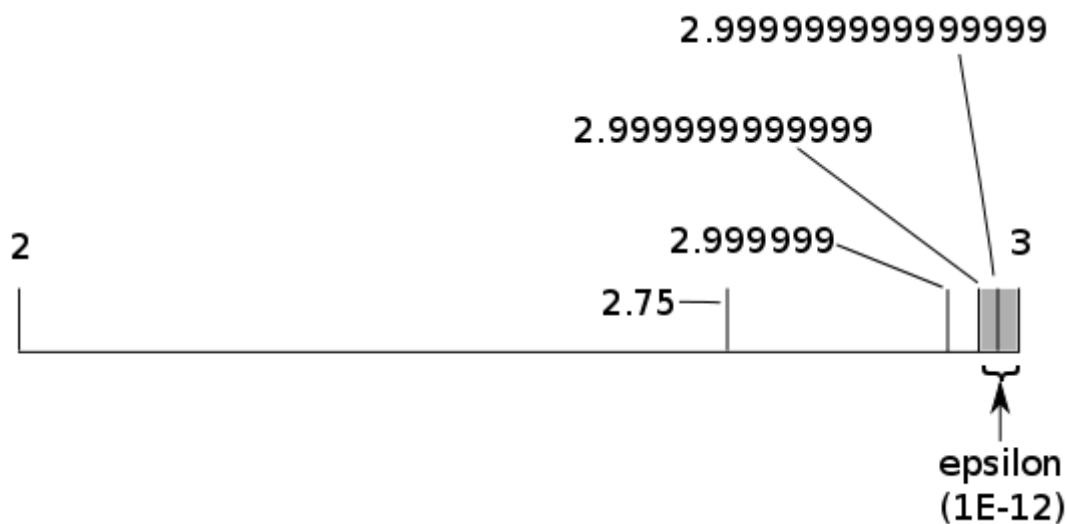
  tot = -2-1e-13;
  in = INT(tot);
  PUT "Because " tot " is within epsilon, returns next higher integer: " in;

RUN;
```

This produces the following output:

```
Because 2.75 is outside of epsilon, returns integer part of number: 2
Because 2.0000000000001 is on boundary of upper epsilon, returns integer part of number: 2
Because -2.0000000000001 is within epsilon, returns next higher integer: -2
```

These results can be visualised on the following number line:



2.999999999999999 falls within the upper epsilon, so the higher integer (3) is returned.
2.999999999999 falls on the boundary of the epsilon, so the integer portion of the value is returned. The other numbers are also not within the epsilon, so the integer of the specified value is returned.

Example – with negative numbers

In this example, the integer part of decimal numbers is returned, unless the fractional part falls within the epsilon of the next lower integer, in which case that integer is returned instead. The result is written to the log.

```
DATA _NULL_;

  tot = -2 - 0.75;
  in = INT(tot);
  PUT "Because " tot "is outside of epsilon, returns integer part of number: " in;

  tot = - 2 - 0.999999999999;
  in = INT(tot);
  PUT "Because " tot 15.12 " is on boundary of upper epsilon, returns integer part
of number: " in;

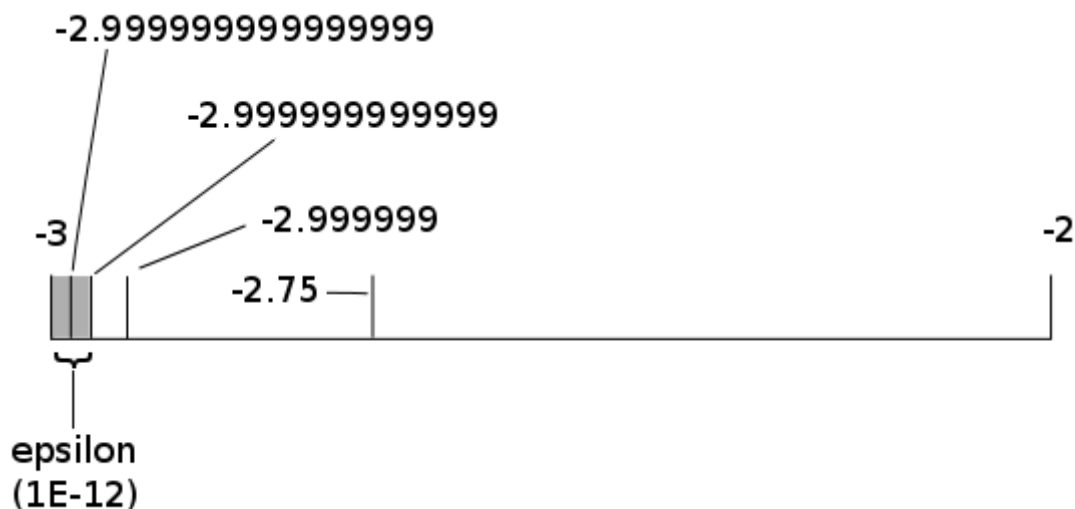
  tot = -2 - 0.999999999999;
  in = INT(tot);
  PUT "Because " tot 16.13 " is within epsilon, returns next higher integer: " in;

RUN;
```

This produces the following output:

```
Because -2.75 is outside epsilon, returns integer part of number: -2
Because -2.999999999999 is on boundary of lower epsilon, returns integer part of
number: -2
Because -2.999999999999 is within epsilon, returns next higher integer: -3
```

These results can be visualised on the following number line:



-2.9999999999999999 falls within the lower epsilon, so the lower integer (-3) is returned.
-2.999999999999 falls on the boundary of the epsilon, so the integer portion of the value is returned (-2). The other numbers are also not within the epsilon, so the integer of the specified value is returned.

INTZ

Returns the integer part of a specified decimal number.

➤ **INTZ** ➤ (*value*) ➤

Return type: Numeric

An integer.

value

Type: Numeric

The number to be rounded.

If you want to return the next lower or higher integer when the fractional part of the number is within a very close range (less than 1E-12) of that integer, use `INT` [↗](#) (page 2131).

Example

In this example, the appropriate integer is returned for rounded decimal numbers. The result is written to the log.

```
DATA _NULL_;

rc = intz(2.99999);
PUT rc = ;

rc = intz(-3.1);
PUT rc = ;

rc = intz(2.999999999999991);
PUT rc = ;

rc = intz(1.000000000000001);
PUT rc = ;

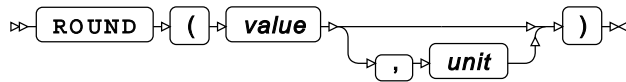
RUN;
```

This produces the following output:

```
rc=2
rc=-3
rc=2
rc=1
```

ROUND

Returns the rounded value of a specified number. If the value is within a very small fraction (an epsilon) of the midpoint of the numbers to which the specified number could be rounded, the result is treated as being at the midpoint and rounded up.



A unit can be specified to which the number will be rounded. The number is rounded to the closest multiple of that unit. The closest multiple will be the higher multiple if the last digit is five or greater, or the lower multiple if the last digit is less than five. For example:

- If the number is 5.25, and the unit is 0.1, the number is rounded up to 5.3
- If the number is 5.4, and the unit is 1, the number is rounded down to 5

Return type: Numeric

The rounded number.

value

Type: Numeric

The number to be rounded.

unit

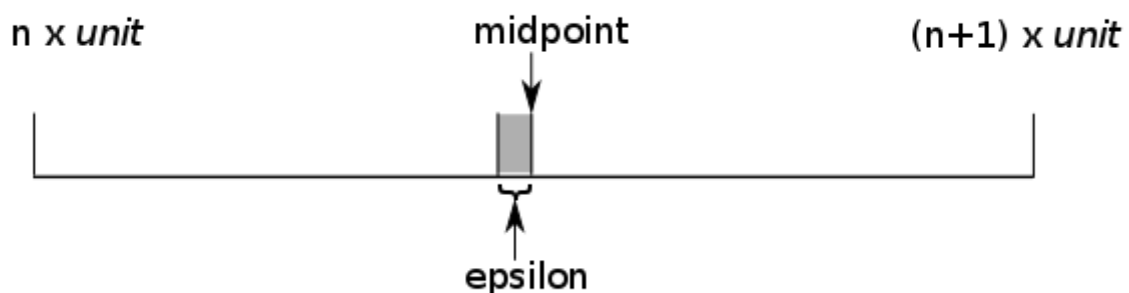
Optional argument

Type: Numeric

The unit to which the number is rounded. By default, this is 1.

Multiples of *unit* are calculated starting from 0 (zero).

ROUND can be visualised as acting on a number line:



In this diagram, *epsilon* indicates a very small fraction to the midpoint between numbers. If the fractional part of a number falls within the epsilon, then the number is treated as being exactly at the midpoint. For example, if *unit* is 0.5, and *value* is 1.2499999999999999, the number falls within the epsilon and is treated as equivalent to 1.25; the number is therefore rounded to 1.5, the next appropriate multiple of the unit.

The fraction at which the epsilon starts depends on the order of magnitude of the specified number. For numbers up to 1000, the epsilon is less than 1E-12. For numbers above 1000, the epsilon is less than 1E-6. For example, if *unit* is 1, then 1000.4999999 is equivalent to 1000.5, and 5.499999999999999 is equivalent to 5.5.

If you want to round to the nearest even multiple of a unit when the specified number is exactly halfway between units, use `ROUNDZ` [\(page 2142\)](#). If you want to round to the nearest even multiple of a unit when the specified number is exactly halfway between units or within an epsilon of halfway, see `ROUNDE` [\(page 2139\)](#).

Basic example

In this example, a number is rounded using the default unit. The result is written to the log.

```
DATA _NULL_;

    rc = ROUND(5.51);
    PUT "Rounds to: " rc;

RUN;
```

This produces the following output:

```
Rounds to: 6
```

Example – specifying a unit

In this example, two numbers are rounded using a specified unit. The result is written to the log.

```
DATA _NULL_;

    rc = ROUND( 5.05, 0.1);
    PUT "Rounds to: " rc;

    rc = ROUND( 5.343, 0.01);
    PUT "Rounds to: " rc;

RUN;
```

This produces the following output:

```
Rounds to: 5.1
Rounds to: 5.34
```

In the first use of the function, the unit specified is 0.1; because the next significant digit is 5 or greater, the next highest multiple of that unit is returned, 5.1. In the second use of the function, the unit specified is 0.01; because the next significant digit is lower than 5, the lower multiple of that unit is returned, 5.34.

Example – specifying a number with a fraction within the epsilon

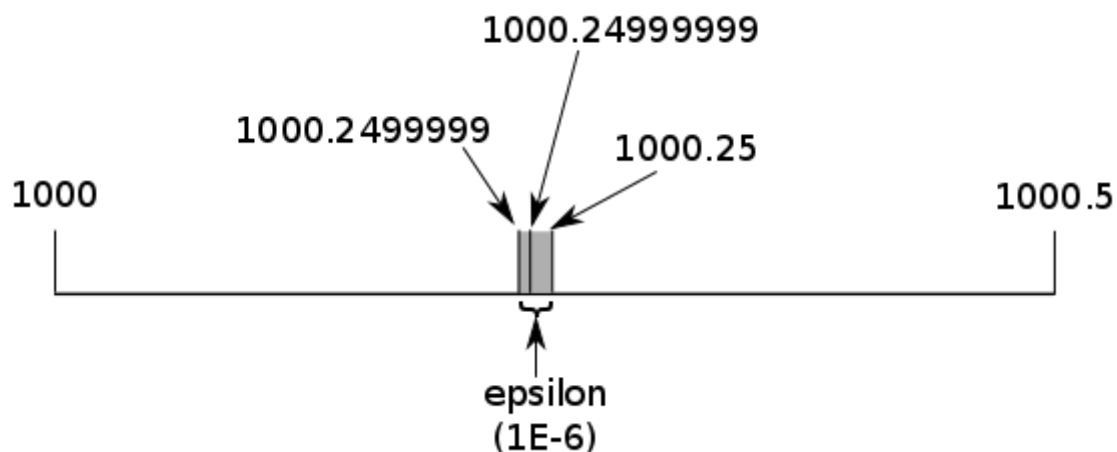
In this example, two numbers are rounded. Both are within the epsilon for rounding. The result is written to the log.

```
DATA _NULL_;  
  
rc = ROUND(1000.24999999, 0.5);  
PUT "Rounds to: " rc;  
  
rc = ROUND(1000.24999999, 0.5);  
PUT "Rounds to: " rc;  
  
RUN;
```

This produces the following output:

```
Rounds to: 1000.5  
Rounds to: 1000
```

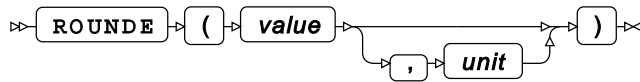
These results can be visualised on the following number line:



The epsilon for numbers above 1000 is 1E-6. The number 1000.24999999 falls within the epsilon, so is treated as 1000.25, and rounded up to the next multiple of 0.5 (1000.5). The number 1000.24999999 is on the boundary of the epsilon, but not within it; it is below 1000.25, so is rounded down to 1000.

ROUNDE

Returns the rounded value of a specified number. If the value falls exactly halfway, or is within a very small fraction (the epsilon) of halfway between numbers to which the specified number could be rounded, the nearest even multiple of the specified unit is returned.



A unit can be specified to which the number will be rounded. The number is rounded to the closest multiple of that unit. By default the unit is 1.

A number is rounded up if greater than halfway between units, or rounded down if less than halfway between units. For example, 5.5 (with a default unit of 1) is rounded up to 6, while 5.4 is rounded down to 5. If the number is 5.25, and the unit is 0.1, the number is rounded up to 5.3.

However, if a number lies exactly between two units, it is rounded to the nearest even multiple of the unit. For example, if the number is:

- 5.5, and the unit is 1, the number is rounded up to 6
- 5.125 and unit is 0.25, the number is rounded down to 5
- 5.15 and the unit is 0.1, the number is rounded up to 5.2.

Because rounding is to the nearest even multiple of the unit, the specified number might be rounded up or down.

A number will be treated as halfway between units if it is within the epsilon of the halfway point.

If you want to round to the nearest even multiple of a unit when the specified number is exactly halfway between units, ignoring the epsilon, use [ROUNDZ](#) (page 2142). If you want to round to the next higher number if the specified number is within an epsilon of halfway, see [ROUND](#) (page 2136) .

Return type: Numeric

The rounded number.

value

Type: Numeric

The number to be rounded.

unit

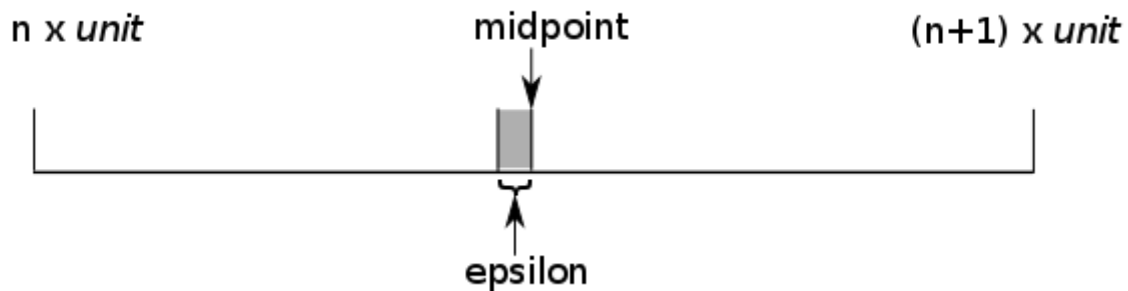
Optional argument

Type: Numeric

The unit to which the number is rounded. By default, this is 1.

Multiples of *unit* are calculated starting from 0 (zero).

How `ROUNDE` behaves if the number falls within the epsilon can be visualised as acting on a number line:



In this diagram, *epsilon* indicates a very small fraction below the midpoint between two numbers $n1$ and $n2$. If the fractional part of a number falls within the epsilon, then the number is treated as being exactly at the midpoint. For example, if *unit* is 0.5, and *value* is 1.7499999999999999, the number falls within the epsilon and is treated as equivalent to 1.75; the number is therefore rounded to 2.0, the next appropriate multiple of the unit, rather than 1.5.

The fraction at which the epsilon starts depends on the order of magnitude of the specified number. For numbers up to 1000, the epsilon is less than 1E-12. For numbers above 1000, the epsilon is less than 1E-6. For example, if *unit* is 1, then 1000.4999999 is equivalent to 1000.5, and 5.499999999999999 is equivalent to 5.5.

Basic example

In this example, two numbers are rounded to the next even multiple of the default unit. The result is written to the log.

```
DATA _NULL_;  
  
re = ROUNDE(3.5);  
PUT "Rounds to: " re;  
  
re = ROUNDE(2.5);  
PUT "Rounds to: " re;  
  
RUN;
```

This produces the following output:

```
Rounds to: 4  
Rounds to: 2
```

The first use of the function, the number is rounded up, and in the second use, the number is rounded down.

Example – specifying a unit

In this example, two numbers are rounded to the next even multiple of the specified unit. The result is written to the log.

```
DATA _NULL_;  
  
    rc = ROUNDE(5.05, 0.1);  
    PUT rc = ;  
  
    rc = ROUND(5.05, 0.1);  
    PUT rc = ;  
  
RUN;
```

This produces the following output:

```
Rounds to: 5  
Rounds to: 5.1  
Rounds to: 5.2
```

In the functions in this example, the unit is set to 0.1. In the first use of `ROUNDE`, the number returned is 5; because 5.05 is exactly halfway between multiples of the unit, the nearest even multiple of the unit (5.0) is returned. Compare this to the result for `ROUND`, where the number returned is 5.1. In this case, because 5.05 is exactly halfway between multiples of the unit, the number returned is the higher multiple of the unit.

In the second use of `ROUNDE`, the number returned is 5.2. 5.149999999999999 is within 1E-12 of 5.15, which is halfway between two multiples of the unit. The nearest even multiple of the unit to which the number can be rounded is therefore 5.2.

Example – specifying a number with a fraction within the epsilon

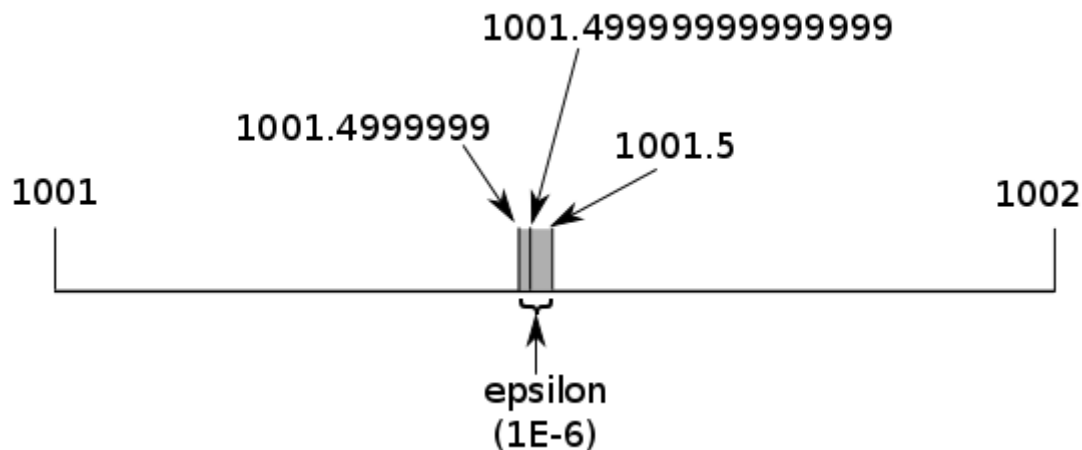
In this example, two numbers are rounded. Both are within the epsilon for rounding. The result is written to the log.

```
DATA _NULL_;  
  
    rc = ROUNDE(1001.4999999999999, 1);  
    PUT "Rounds to: " rc;  
  
    rc = ROUNDE(1001.49999, 1);  
    PUT "Rounds to: " rc;  
  
RUN;
```

This produces the following output:

```
Rounds to: 1002  
Rounds to: 1001
```

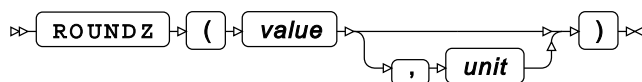
These results can be visualised on the following number line:



The epsilon for numbers above 1000 is 1E-6. The number 1001.4999999999999 falls within the epsilon, so is treated as 1001.5, and rounded to the nearest even multiple of 0.5 (1002). The number 1001.4999999 is on the boundary of the epsilon, but not within it; it is below 1000.5, so is rounded down to 1001.

ROUNDZ

Returns the rounded value of a specified number. If the value falls exactly halfway between numbers to which the specified number could be rounded, the nearest even multiple of the specified unit is returned.



A unit can be specified to which the number will be rounded. The number is rounded to the closest multiple of that unit. The closest multiple will be the higher multiple if the last digit is five or greater, or the lower multiple if the last digit is less than five. For example:

- If the number is 5.25, and the unit is 0.1, the number is rounded up to 5.3
- If the number is 5.4, and the unit is 1, the number is rounded down to 5

If rounding results in a value that is exactly halfway between even multiples of rounding units, the nearest even multiple that matches the unit is returned.

Return type: Numeric

The rounded number.

value

Type: Numeric

The number to be rounded.

unit

Optional argument

Type: Numeric

The unit to which the number is rounded. By default, this is 1.

Multiples of *unit* are calculated starting from 0 (zero).

Basic example

In this example, two numbers are rounded. The result is written to the log.

```
DATA _NULL_;

rc = ROUNDZ(5.17);
PUT rc = ;

rc = ROUNDZ(5.59, 0.5);
PUT rc = ;

RUN;
```

This produces the following output:

```
rc=5
rc=5.5
```

In the first use of the function, no unit is specified, so the default unit is used (1); `ROUNDZ`, the number returned is 5. In the second use of the function, because 5.17 is closer to 5.2, the next multiple of the unit, than it is to 5.1, that is the value returned.. In the second use of `ROUNDZ`, the number returned is 5.1; because 5.149999999999999 is less than halfway between units, the lower multiple of the unit is returned.

Example – rounding to even multiple

In this example, two numbers fall exactly between multiples of the specified unit, and are rounded to the closest even multiple of the unit. The result is written to the log.

```
DATA _NULL_;

rc = ROUNDZ(5.25, 0.5);
PUT rc = ;

rc = ROUNDZ(5.75, 0.5);
PUT rc = ;

RUN;
```

This produces the following output:

```
rc=5
rc=6
```

In both functions in this example, the unit is set to 0.5. In the first use of `ROUNDZ`, the number returned is 5 rather than 5.5, because 5 is an even multiple of 0.5, while 5.5 is an odd multiple of the unit. In the second use of `ROUNDZ`, the number returned is 6; because 5.75 is halfway between multiples of the unit, the nearest even multiple of the unit is returned, which in this case is 6.

TRUNC

Returns the number that results from truncating the floating point representation of a specified number to a defined number of bytes.

➤ `TRUNC` ➤ `(value , length)` ➤

A number is represented in a computer using some form of floating-point architecture; for example, the IEEE 754 standard, or the IBM hexadecimal floating-point format. These architectures ensure that numbers are represented as accurately as possible, and that operations on them return accurate and expected results.

In floating-point architectures, numbers are represented by binary numbers in a normalised scientific notation, consisting of a sign bit, an exponent, and a mantissa. For example, the number 5.15 would be represented using the IEEE 754 standard as:

```
0 10000001 01001001100110011001101
```

where, reading from left to right:

- 0 is the sign bit (positive)
- 10000001 is the exponent
- 01001001100110011001101 is the mantissa

In this case, the number is represented as a single precision number and consists of 32 bits. A double precision number would consist of 64 bits, and the exponent would then consist of 11 bits, and the mantissa of 52 bits.

This function truncates the binary representation of a number by a specified number of bytes, starting from the right; that is, the mantissa is truncated. The truncated portion is padded with zeros.

The IEEE floating-point standard can be found at [IEEE Standard for Binary Floating-Point Arithmetic](#). Information on IBM hexadecimal floating point can be found at [Hexadecimal floating-point literals](#).

Return type: Numeric

The truncated number.

value

Type: Numeric

The number to be truncated.

length**Type:** Numeric

The number of bytes to which to truncate the number. The value can be:

- 2 to 8 on z/OS
- 3 to 8 on all other operating systems

If you specify any other value, an error message and a missing value is returned.

For example, as noted above, the single precision representation of 5.15 is represented, using the IEEE 754 standard, as:

```
010000000101001001100110011001101
```

If you truncate to the third byte, then the result would be:

```
010000000101001001100110000000000
```

as the truncated digits are padded with zeros. This floating point representation would then be returned by the function as the equivalent decimal, 5.1494140625 (using default formats).

Example

In this example, two numbers are truncated to the specified number of bytes. The result is written to the log.

```
DATA _NULL_;

  rc = TRUNC(5.15, 3);
  PUT rc = ;

  rc = TRUNC(5.15, 4);
  PUT rc = ;

  rc = TRUNC(5.15, 1);
  PUT rc = ;

RUN;
```

This produces the following output:

```
rc=5.1494140625
rc=5.1499977112
NOTE: Argument 2 to function TRUNC at line 2333 column 8 is invalid
rc=.
```

In the third use of the function, an invalid value for *length* is specified; this causes an error message to be generated.

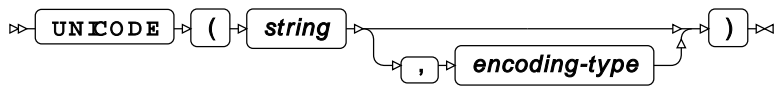
Unicode functions

Convert strings and characters to Unicode format.

UNICODE	2146
Returns a string of text converted from Unicode.	
UNICODEC	2148
Returns a string in which characters of a source string are replaced by Unicode code points of a specified format.	
UNICODELEN	2150
Returns the length of a string encoded in Unicode code points.	

UNICODE

Returns a string of text converted from Unicode.



Converts a string of Unicode code points to its text equivalent. You can specify the encoding type (such as UTF-8, UTF-16, or UCS-4) to which the code points belong.

Return type: Character

string

Type: Character

The string of code points to be converted.

encoding-type

Optional argument

Type: Character

The Unicode encoding of the code points comprising *source*. This can be:

UTF8	Convert <i>source</i> to UTF-8.
UCS2	Convert <i>source</i> to UTF-16.
UTF16	Convert <i>source</i> to UTF-16.
UCS2B	Convert <i>source</i> to UTF-16BE.
UTF16B	Convert <i>source</i> to UTF-16BE.
UCS2L	Convert <i>source</i> to UTF-16LE.
UTF16L	Convert <i>source</i> to UTF-16LE.

UCS4	Convert <i>source</i> to UTF-32LE.
UCS4B	Convert <i>source</i> to UTF-32BE.
UCS4L	Convert <i>source</i> to UTF-3LBE.
ESC	Convert Unicode code points specified with the escape (\) character to the equivalent character in the project character set; for example, \u5927 is converted to the letter a. This is the default.
NCR	Convert Unicode code points specified using numeric character reference format to the equivalent character in the project character set; for example: 大 is converted to the letter a.
PAREN	Convert Unicode code points specified using parentheses (<>) to the equivalent character in the project character set; for example: <u5927> is converted to the letter a.

A code point can only be converted to a character if the current character set for the session contains that code point.

Example

In this example, the function is used to convert hexadecimal strings to UTF-8 and UTF-16 encodings. The result is written to the log.

```
DATA _NULL_;
  result1 = UNICODE('004C006F006F006B0021203A'x, 'UTF16');
  result2 = UNICODE('004C006F006F006B0021203A'x, 'UTF8');
  result3 = UNICODE('\u004C\u006F\u006F\u006B\u0021\u203A', 'ESC');
  PUT 'Conversion to UTF-16:           ' result1;
  PUT 'Conversion to UTF-8:           ' result2;
  PUT 'Conversion from escaped Unicode: ' result3;
RUN;
```

In WPS on Windows 10 with the default character set (CP1252), this produces the following output:

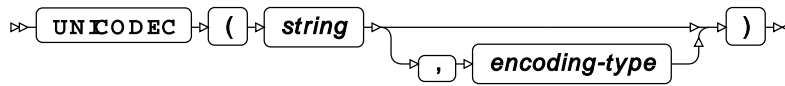
```
Conversion to UTF-16:           Look!>
Conversion to UTF-8:           L o o k ! :
Conversion from escaped Unicode: Look!>
```

The UTF-8 encoding has alternate unprintable characters (determined by the hexadecimal values '00'x and '20'x), because in UTF-8 a character might be represented by one, two or more bytes. For example, the string begins with '004C'x, and in UTF-8, '00'x is the NUL character, and '4C'x is the L character. In UTF-16, however, a character is represented by two two-byte groups; therefore, the same '004C'x is the L character. Similarly the string ends with the hexadecimal number '203A'x. In UTF-8, this represents two characters, '20'x, the space character, and '3A'x, the colon (:). In UTF-16, the hexadecimal number '203A'x is the single right-pointing angle quotation mark.

In the third example of the function, the characters are Unicode codepoints identified by the \u combination, and these are returned as characters from the character set of Workbench.

UNICODEC

Returns a string in which characters of a source string are replaced by Unicode code points of a specified format.



Although this function can be used to replace any character with a Unicode code point, it is perhaps most useful for replacing characters that cannot be represented in the session encoding. You can specify the code set (for example, UTF-8, UTF-16, or UCS-4) in which you want code points returned. You can alternatively specify that characters are converted to a code point representation.

Return type: Character

source

Type: Character

The string to be converted.

encoding-type

Optional argument

Type: Character

The Unicode format in which the characters in *source* are returned. The following options are available:

UTF8	Convert <i>source</i> to UTF-8.
UCS2	Convert <i>source</i> to UTF-16.
UTF16	Convert <i>source</i> to UTF-16.
UCS2B	Convert <i>source</i> to UTF-16BE.
UTF16B	Convert <i>source</i> to UTF-16BE.
UCS2L	Convert <i>source</i> to UTF-16LE.
UTF16L	Convert <i>source</i> to UTF-16LE.
UCS4	Convert <i>source</i> to UTF-32LE.
UCS4B	Convert <i>source</i> to UTF-32BE.
UCS4L	Convert <i>source</i> to UTF-32LE.
ESC	Convert characters other than ASCII printable characters to Unicode code points specified with the escape (<code>\</code>) character. This is the default. For example, <code>a</code> is represented as <code>a</code> , but <code>À</code> is represented as <code>\u00C0</code> .
NCR	Convert characters other than ASCII printable characters to Unicode code points specified with the numeric character reference format. For example, <code>a</code> is represented as <code>a</code> , but <code>À</code> is represented as <code>&#192;</code> .

PAREN	Convert characters other than ASCII printable characters to Unicode code points specified using parentheses (<>). For example, a is represented as a, but À is represented as <u00C0>.
-------	--

If a codepoint is specified, it will only be converted to a character if the session encoding contains that codepoint.

The formatted length of the value returned is the same length as *source*. To ensure the returned value is output in full, specify a suitable length.

Example – using code point representation

In this example, encoded strings are returned in the specified format. The result is written to the log.

```
DATA NULL;

    LENGTH result $10;

    result = UNICODDEC('A大');
    PUT "ESC returns:  " result;

    result = UNICODDEC('A大', "NCR");
    PUT "NCR returns:  " result ;

    result = UNICODDEC('A大', "PAREN");
    PUT "PAREN returns: " result;

RUN;
```

This produces the following output:

```
ESC returns:  A\u5927
NCR returns:  A&#22823;
PAREN returns: A<u5927>
```

The character A is an ASCII printable character, so is not converted to a Unicode code point reference.

Example – using a different UTF encoding

In this example, strings are returned in the specified UTF encoding. The `UNICODE` function is first used to create a character using escaped Unicode format. The result is written to the log.

```
DATA NULL;

    x = UNICODE('\u03d7');

    result1 = UNICODDEC(x, "UTF8");
    put result1 $hex.;

    result2 = UNICODDEC(x, "UTF16");
    put result2 $hex.;
    put x;

RUN;
```

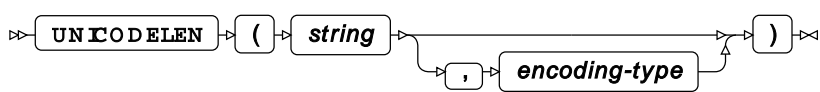
This produces the following output:

```
CF9720202020
FFED7032000
⌘
```

Note:
In this example, the character ⌘ can be displayed in the Workbench session encoding.

UNICODELEN

Returns the length of a string encoded in Unicode code points.



Find the length of a string in a specified Unicode encoding, and return the result. You can specify that the string is comprised of code points in various encodings.

Return type: Numeric

string

Type: Character

The string whose length is to be found.

encoding-type

Optional argument

Type: Character

The Unicode encoding for *source*. This can be:

UTF8	UTF-8.
UCS2	UTF-16.
UTF16	UTF-16.
UCS2B	UTF-16BE.
UTF16B	UTF-16BE.
UCS2L	UTF-16LE.
UTF16L	UTF-16LE.
UCS4	UTF-32LE.
UCS4B	UTF-32BE.
UCS4L	UTF-32LBE.

ESC	Unicode codepoints specified with the escape (\) character; for example, \u5927 (the letter a). This is the default.
NCR	Unicode codepoints specified as numeric character references; for example: 大 (the letter a).
PAREN	Unicode codepoints specified using parentheses (<>); for example: <u5927> (the letter a).

Example

In this example, the function finds the length of a string in UTF-8 and UTF-16 encoding. The result is written to the log.

```
DATA _NULL_;
  result1 = UNICODELEN('004C006F006F006B0021203A'x, 'UTF8');
  result2 = UNICODELEN('004C006F006F006B0021203A'x, 'UCS2B');
  result3 = UNICODELEN('\u004C\u006F\u006F\u006B\u0021\u203A', 'ESC');
  PUT result1;
  PUT result2;
  PUT result3;
RUN;
```

In WPS on Windows 10 with the default character set (CP1252), this example produces the following output:

```
12
6
6
```

The UTF-8 encoding has alternate unprintable characters (determined by the hexadecimal values '00'x and '20'x), because in UTF-8 a character might be represented by one, two or more bytes. For example, the string begins with '004C'x, and in UTF-8, '00'x is the NUL character, and '4C'x is the L character. In UTF-8, the string is L o o k ! : and is therefore twelve characters long. In UTF-16, however, a character is represented by two two-byte groups; therefore, the same '004C'x is the L character. Similarly, at the end of the string the hexadecimal number '1203'x represents space ('20'x), and colon (:) in UTF-8. In UTF-16, the hexadecimal number '203A'x represents the single right-pointing angle quotation mark (>). In UTF-16, the string is Look! >, and is six characters long.

In the third result, the number of Unicode characters identified by the escape character (\) are counted; the string is therefore six characters long, and 6 is returned as the length of the string.

Value formatting and assignment functions

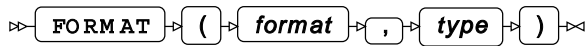
Format and assign data to variables.

Data can be formatted and assigned to a variable. The data can be formatted for input (using an informat) or for output (using a format). The type (numeric or character) of formats can also be checked. The formats or informats can be system-defined or user-defined (for example, through PROC FORMAT).

FORMAT ↗	2152
Return a value that indicates whether a specified format is known and valid, and, if so, whether it is a character or numeric format.	
INFORMAT ↗	2154
Return a value that indicates whether a specified informat is known and valid, and, if so, whether it is a character or numeric informat.	
INPUT ↗	2156
Returns the formatted value of a variable to which a specified informat has been applied.	
INPUTC ↗	2157
Returns a value that has been formatted using the specified informat.	
INPUTN ↗	2158
Returns a numeric value from a string, that is read in using a numeric informat.	
PUT ↗	2160
Returns the formatted value of a variable to which a specified format has been applied.	
PUTC ↗	2161
Returns value that has been formatted using the specified format.	
PUTN ↗	2162
Returns a number in a specified numeric format.	

FORMAT

Return a value that indicates whether a specified format is known and valid, and, if so, whether it is a character or numeric format.



Return type: Numeric

- 1 The format is known and valid, and is of the specified type
- 0 The format is unknown, invalid, or does not have the specified type.

format

Type: Character

The name of a format. The format must be specified as it is defined in [Formats](#) (page 373); it must contain a period. If it does not, it is not recognised as a format. For example, \$HEX is not a valid format, \$HEX. is. The specified format can include width and decimal if required; for example, BEST. and BEST4.2 are both numeric formats.

You can also specify a user-defined format.

type

Specifies for which type, numeric or character, the format is checked.

"C"

Checks whether the specified format is a character format.

"N"

Checks whether the specified format is a numeric format.

Basic example

In this example, various formats are checked for validity and type. The result is written to the log.

```
DATA _NULL_;
  len = 12;
  INPUT fy $VARYING12. len;

  IF FORMAT(fy,"n") EQ 1 OR INFORMAT(strip(fy),"c") EQ 1 THEN
    DO;
      IF FORMAT(fy,"n") EQ 1 THEN PUT fy $CHAR11. "is a valid numeric format";
      IF FORMAT(fy,"c") EQ 1 THEN PUT fy $CHAR11. "is a valid character format";
    END;
  ELSE PUT fy $CHAR11. "is an invalid format";

DATALINES;
$HEX32768.
$HEX32767.
D12.5
HEXA.
$ASCI112.
;
```

This produces the following output:

```
$HEX32768. is an invalid format
$HEX32767. is a valid character format
D12.5      is a valid numeric format
HEXA.      is an invalid format
$ASCI112.  is a valid character format
```

The formats are valid, apart from `HEXA.`, which does not exist in WPS and is not a user-defined format, and `$HEX32768.`, the width of which exceeds the maximum width for the format.

Example – user-defined informat

In this example, an informat is created using `PROC FORMAT`; this informat is then specified to the `INFORMAT` function. The result is written to the log.

```
PROC FORMAT;
  VALUE ftype
    1 = "commercial"
    2 = "self-funding"
;
RUN;

DATA _NULL_;

  isc = FORMAT("ftype.", "n");
  rc = IFC(isc, "the format is a numeric format", "the format is not a numeric
format");
  PUT "The return code is " isc "so " rc;

RUN;
```

This produces the following output:

```
The return code is 1 so the format is a numeric format
```

INFORMAT

Return a value that indicates whether a specified informat is known and valid, and, if so, whether it is a character or numeric informat.

```
INFORMAT ( informat , type )
```

Return type: Numeric

- 1 The informat is known and valid, and is of the specified type
- 0 The informat is unknown, invalid, or does not have the specified type.

informat

Type: Character

The name of an informat. The informat must be specified as it is defined in [Informats](#) (page 538); it must contain a period. If it does not, it is not recognised as an informat. For example, `$PHEX` is not a valid informat, `$PHEX.` is. The specified informat can include width and decimal if required; for example, `BEST.` and `BEST4.2` are both numeric informats.

You can also specify a user-defined informat.

type

Specifies for which type, numeric or character, the informat is checked.

"C"

Checks whether the specified informat is a character informat.

"N"

Checks whether the specified informat is a numeric informat.

Basic example

In this example, various informats are checked for validity and type. The result is written to the log.

```
DATA _NULL_;
  len = 12;
  INPUT infy $VARYING12. len;

  if INFORMAT(infy,"n") EQ 1 OR INFORMAT(strip(infy),"c") EQ 1 then
    do;
      IF INFORMAT(infy,"n") EQ 1 THEN PUT infy $CHAR11. "is a valid numeric format";
      IF INFORMAT(infy,"c") EQ 1 THEN PUT infy $CHAR11. "is a valid character
format";
    END;
    ELSE PUT infy $CHAR11. "is an invalid format";

  DATALINES;
  $HEX32768.
  $HEX32767.
  HEX4.5
  HEXA.
  $ASCII12.
  ;
```

This produces the following output:

```
$HEX32768. is an invalid format
$HEX32767. is a valid character format
HEX4.5      is a valid numeric format
HEXA.       is an invalid format
$ASCII12.   is a valid character format
```

The informats are valid, apart from `HEXA.`, which does not exist in WPS and is not a user-defined informat, and `$HEX32768.`, the width of which exceeds the maximum width for the informat.

Example – user-defined informat

In this example, an informat is created using `PROC FORMAT`; this informat is then specified to the `INFORMAT` function. The result is written to the log.

```
PROC FORMAT;
  INVALUE $ftype
    'co' = 'commercial'
    'sf' = 'self-funded'
;
RUN;

DATA _NULL_;

  isc = INFORMAT("$ftype.", "c");
  fi = IFC(isc, "The informat is a character informat", "The informat is not a
character informat");
  PUT fi;

RUN;
```

This produces the following output:

```
The informat is a character informat
```

INPUT

Returns the formatted value of a variable to which a specified informat has been applied.

➤ **INPUT** ➤ (➤ *string* ➤ , ➤ *informat* ➤) ➤

Return type: Character

Formatted input.

string

Type: Character

The input variable to be formatted.

informat

Type: Informat

An informat, as described in the section *Informats*.

Example

In this example, the function is used to apply informats to data. The result is written to the log.

```
DATA _NULL_;

  INPUT prod $ price;

  IF _N_ EQ 2 THEN DO;
    f1 = INPUT(prod, $REVERJ10.);
    f2 = INPUT(price, BEST4.2);
    PUT f1 f2;
  END;
  ELSE DO;
    f1 = INPUT(prod, $F3.);
    f2 = INPUT(price, 6.3);
    PUT f1 f2;
  END;

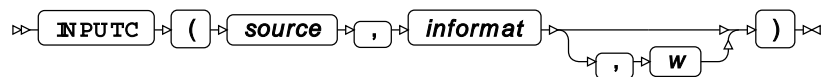
DATALINES;
Tea 1002
Cabinet 200506
Coffee 3713
```

This produces the following output:

```
Tea 1.002
tenibaC 2005.06
Cof 3.713
```

INPUTC

Returns a value that has been formatted using the specified informat.



Return type: Character

source

Type: Character

The source value to be formatted.

informat

Type: Character

The informat to apply *source*. The format must be enclosed in quotation marks; for example, '\$CHAR5.'. You do not have to enter either the leading \$ or the terminating period. of the format; for example, '\$CHAR5.' and 'CHAR5' are equivalent.

For information on formats, see [Formats](#) (page 373).

w

Optional argument

Type: Numeric

The width to apply to the format. This value overrides any width specified to the format in *format*. For example, if you specify `var = INPUTC('sometext', '$CHAR5', 7)`, the value returned will be `sometex`.

Example

In this example, a string and a numeric variable are used with the function. The result is written to the log.

```
DATA _NULL_;

  str = 'Hello world';
  a = 1;

  out = INPUTC(str, '10.', 5);
  PUT 'str: ' out;

  out = INPUTC(a, '10.', 5);
  PUT 'num:  out;

RUN;
```

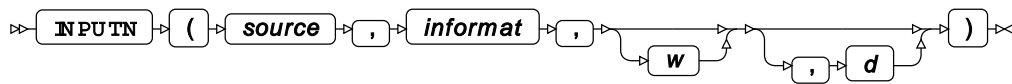
This produces the following output:

```
str: Hello
num:
```

The string is formatted; however, the function cannot convert the numerical variable to a character string, so a string of the specified length that contains only spaces is returned.

INPUTN

Returns a numeric value from a string, that is read in using a numeric informat.



Return type: Numeric

source

Type: Character

The input string.

informat

Type: Character

The informat applied to the source.

w

Optional argument

Type: Numeric

The width to apply to the informat.

d

Optional argument

Type: Numeric

The number of decimal digits to apply to the informat.

Basic example

In this example, a numerical string and a numerical variable are used with the function. The result is written to the log.

```
DATA _NULL_;  
  str = "12345678910";  
  a = 1;  
  
  out = INPUTN(str, '8.2', 5, 3);  
  PUT "str: " out;  
  
  out = INPUTN(a, '8.2', 5);  
  PUT "num: " out;  
RUN;
```

This produces the following output:

```
str: 12.345  
num: .
```

The numerical string is processed, however the function is unable to convert the numerical variable to a numerical string for output.

Example – removing symbols

In this example, the specified informat does what is expected, removing the '\$' and ',' symbols and returns the number. The result is written to the log.

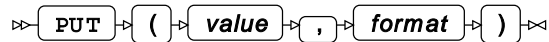
```
DATA _NULL_;  
  str = "$1,35907";  
  out = INPUTN(str, 'dollar10.2');  
  PUT "str: " out;  
RUN;
```

This produces the following output:

```
str: 135907
```

PUT

Returns the formatted value of a variable to which a specified format has been applied.



Return type: Character

Formatted variable. Numbers are formatted and returned as characters.

value

Type: Character or numeric value

The value to be formatted.

format

Type: Format

A format. For information on formats, see [Formats](#) (page 373).

Example

In this example, the function is used to apply formats to variables. The result is written to the log.

```
DATA _NULL_;

INPUT prod $ price;

IF _N_ EQ 2 THEN DO;
    f1 = PUT(prod, $REVERJ10.);
    f2 = PUT(price, BEST4.2);
    PUT f1 f2;
END;
ELSE DO;
    f1 = PUT(prod, $F3.);
    f2 = PUT(price, 6.3);
    PUT f1 f2;
END;

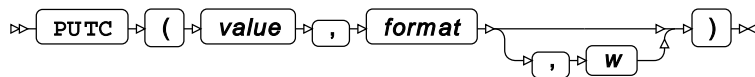
DATALINES;
Tea 1002
Cabinet 200506
Coffee 3713
```

This produces the following output:

```
Tea 1002
tenibaC 2E5
Cof 3713
```

PUTC

Returns value that has been formatted using the specified format.



Return type: Character

The specified character format.

value

Type: Character

The value to be formatted.

format

Type: Character

The format applied to the character. The format must be enclosed in quotation marks; for example, '\$CHAR5.'. You do not have to enter either the leading \$ or the terminating period. of the format; for example, '\$CHAR5.' and 'CHAR5' are equivalent.

For information on formats, see [Formats](#) (page 373).

w

Optional argument

Type: Numeric

The width to apply to the format. This value overrides any width specified to the format in *format*. For example, if you specify `var = PUTC('sometext', '$CHAR5.', 7)`, the value returned will be `sometex`.

Example

In this example, the function is used to format two strings. The result is written to the log.

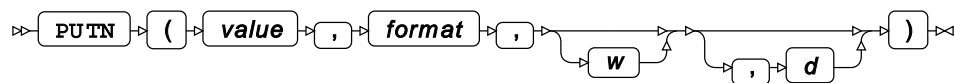
```
DATA _NULL_;  
  
x='AbCDeFGHi';  
  
var=PUTC(x, '$UPCASE');  
PUT 'The formatted string is: ' var;  
  
var=PUTC(x, '$5.', 7);  
PUT 'The formatted string is: ' var;  
  
RUN;
```

This produces the following output:

```
The formatted string is: ABCDEFGHI  
The formatted string is: AbCDeFG
```

PUTN

Returns a number in a specified numeric format.



Return type: Character

The specific numeric format.

value

Type: Numeric

The variable.

format

Type: Character

The format to be applied to the number.

w

Optional argument

Type: Numeric

The width of the format.

d

Optional argument

Type: Numeric

The number of digits to apply to the format.

Example

In this example, in the first use of the function, it is configured to display only two decimal places. In the second use of the function, a binary format is specified. The result is written to the log.

```
DATA _NULL_;
  y=142.00039;
  col1=PUTN(y, '6.0', 6, 2);
  col2=PUTN(y, 'BINARY.', 8, 0);
  PUT col1 col2;
RUN;
```

This produces the following output:

```
142.00 10001110
```

Variable information functions and CALL routines

These functions return information about variables assigned in the DATA step, either explicitly or from a dataset.

When a function-name is appended by 'X', variable names should be entered enclosed by quotes.

VARRAY ↗	2165
Returns a value indicating whether it is the name of an array variable.	
VARRAYX ↗	2166
Returns a value indicating whether an array exists when using the specified quoted variable name.	
VFORMAT ↗	2167
Returns the format for the specified variable name.	
VFORMATD ↗	2168
Returns the digits of the format for a specified variable name.	
VFORMATDX ↗	2169
Returns the digits of the format for a specified quoted variable name.	
VFORMATN ↗	2171
Returns the format name for a specified variable name.	
VFORMATNX ↗	2172
Returns the format name for a specified quoted variable name.	

VFORMATW ↗	2174
Returns the width of the format for the specified variable name.	
VFORMATWX ↗	2176
Returns the width of the format for the specified quoted variable name.	
VFORMATX ↗	2178
Returns the format for the specified quoted variable name.	
VINARRAY ↗	2180
Returns a value indicating whether the specified variable name is in an array.	
VINARRAYX ↗	2181
Returns a value indicating whether the specified quoted variable name is in an array.	
VINFORMAT ↗	2182
Returns the informat for a specified variable name.	
VINFORMATD ↗	2183
Returns the digits of the informat for the specified variable name.	
VINFORMATDX ↗	2184
Returns the digits of the informat for the specified quoted variable name.	
VINFORMATN ↗	2185
Returns the informat name for a specified variable name.	
VINFORMATNX ↗	2186
Returns the informat name for a specified quoted variable name.	
VINFORMATW ↗	2188
Returns the width of the informat for the specified variable name.	
VINFORMATWX ↗	2189
Returns the width of the informat for the specified quoted variable name.	
VINFORMATX ↗	2191
Returns the informat for a specified quoted variable name.	
VLABEL ↗	2192
Returns the label of the specified variable name.	
VLABELX ↗	2194
Returns the label of the specified quoted variable name.	
VLENGTH ↗	2196
Returns the length of the specified variable name.	
VLENGTHX ↗	2198
Returns the length of the specified quoted variable name.	
VNAME ↗	2199
Returns the name of a specified variable name.	
VNAMEX ↗	2200
Returns the name of a specified quoted variable name.	
VTRANSCODE ↗	2202
Returns a value indicating whether a specified variable name can be transcoded.	

VTRANSCODEX ↗	2203
Returns a value indicating whether a specified quoted variable name can be transcoded.	
VTYPE ↗	2204
Returns the type of data associated with the specified variable name. This will be N for numeric or C for character.	
VTYPEX ↗	2206
Returns the type of data associated with the specified quoted variable name. This will be N for numeric or C for character.	
VVALUE ↗	2208
Returns the formatted value of the specified variable name.	
VVALUEX ↗	2209
Returns the formatted value of the specified quoted variable name.	
CALL LABEL ↗	2211
Returns the label for a specified variable name. The label is returned as an argument in the CALL routine.	
CALL VNAME ↗	2212
Returns the name for a specified variable name. The name is returned as an argument in the CALL routine.	
CALL VNEXT ↗	2214
Returns the variable names from a DATA step.	

VARRAY

Returns a value indicating whether it is the name of an array variable.



Return type: Numeric

1 if the variable name is an array, 0 (zero) otherwise.

variable-name

Type: Var

The variable name.

Example

In this example, the function is used twice with variables. The result is written to the log.

```
DATA _NULL_;  
  ARRAY nums(5) (15 21 307 42 55);  
  i = 10;  
  
  var = VARRAY(nums);  
  if var = 1 THEN PUT "The nums variable name is an array: " var;  
  ELSE PUT "The nums variable name is not an array " var;  
  
  pax = VARRAY(i);  
  if pax = 1 THEN PUT "The i variable name is an array: " pax;  
  ELSE PUT "The i variable name is not an array: " pax;  
RUN;
```

This produces the following output:

```
The nums variable name is an array: 1  
The i variable name is not an array: 0
```

VARRAYX

Returns a value indicating whether an array exists when using the specified quoted variable name.

⇒ **VARRAYX** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

1 if the variable name is an array, 0 (zero) otherwise.

variable-name

Type: Character

The quoted variable name.

Example

In this example, the function indicates that a specified variable name is an array. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (15 27 39 42 500);
  i = 10;

  var = VARRAYX("nums");
  if var = 1 THEN PUT "nums is a name of an array: " var;
  ELSE PUT "nums is not a name of an array: " var;

  pax = VARRAYX("i");
  if pax = 1 THEN PUT "i is a name of an array: " pax;
  ELSE PUT "i is not a name of an array: " pax;
RUN;
```

This produces the following output:

```
nums is a name of an array: 1
i is not a name of an array: 0
```

Example

In this example, the *nums* variable is then assigned to a second variable *abc*. The function indicates that a specified variable name is an array. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (10 25 3 42 55);
  i = 10;

  abc = "nums";

  var = VARRAYX(abc);
  if var = 1 THEN PUT "abc is a name of an array: " var;
  ELSE PUT "abc is not a name of an array: " var;

  pax = VARRAYX("i");
  if pax = 1 THEN PUT "i is a name of an array: " pax;
  ELSE PUT "i is not a name of an array: " pax;
RUN;
```

This produces the following output:

```
abc is a name of an array: 1
i is not a name of an array: 0
```

VFORMAT

Returns the format for the specified variable name.

→ **VFORMAT** → (→ *variable-name* →) →

Return type: Character

The format for the variable.

variable-name

Type: Var

The variable name.

Example

In this example, formats for variables are returned. Some of these variables have default formats, while others have formats assigned. The result is written to the log.

```
DATA _NULL_;  
  i = 10;  
  FORMAT p21 $8.;  
  p21 = "Twenty-One-Today";  
  dp = 1.7E+308;  
  FORMAT date Date9.;  
  date = "19-MAY-2017"D;  
  
  var = VFORMAT(i);  
  PUT "The format for the i variable is: " var;  
  pax = VFORMAT(p21);  
  PUT "The format for the p21 variable is: " pax;  
  zar = VFORMAT(dp);  
  PUT "The format for the dp variable is: " zar;  
  cur = VFORMAT(date);  
  PUT "The format for the date variable is: " cur;  
RUN;
```

This produces the following output:

```
The format for the i variable is: BEST12.  
The format for the p21 variable is: $8.  
The format for the dp variable is: BEST12.  
The format for the date variable is: DATE9.
```

VFORMATD

Returns the digits of the format for a specified variable name.

⇒ **VFORMATD** ⇒ (⇒ ***variable-name*** ⇒) ⇒

Return type: Numeric

variable-name

Type: Var

The variable name.

Example – using a dataset

In this example, digits of the decimal format for each variable are returned from a dataset. It is known that there are four named variables in a dataset, but the digits of the decimal format for each variable are unknown. The result is written to the log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;

  SET mydir.VARFMAT (OBS=1);

  var_i = VFORMATD(i);
  PUT "The number of digits for the i variable is: " var_i;

  var_sp = VFORMATD(sp);
  PUT "The number of digits for the sp variable is: " var_sp;

  var_dp = VFORMATD(dp);
  PUT "The number of digits for the dp variable is: " var_dp;

  var_frac = VFORMATD(fraction);
  PUT "The number of digits for the fraction variable is: " var_frac;
RUN;
```

This produces the following output:

```
The number of digits for the i variable is: 2
The number of digits for the sp variable is: 3
The number of digits for the dp variable is: 4
The number of digits for the fraction variable is: 3
```

The properties of the VARFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
dp	number	8	8.4		
fraction	number	8	7.3		
i	number	8	BEST3.2		
sp	number	8	4.3		

VFORMATDX

Returns the digits of the format for a specified quoted variable name.

⇒ **VFORMATDX** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a format and value assigned to it. The number of digits for the format of the variable are then returned. The result is written to a log.

```
DATA _NULL_;  
  FORMAT a F5.2;  
  a = 2.554;  
  
  fmtv = VFORMATDX("a");  
  PUT "The number of digits is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The number of digits is: 2
```

Example

In this example, a variable has a format and value assigned to it. The variable is then assigned to a second variable. The number of digits for the format of the first variable are then returned. The result is written to a log.

```
DATA _NULL_;  
  
  FORMAT a F5.2;  
  a = 2.554;  
  b = "a";  
  
  fmtv = VFORMATDX(b);  
  PUT "The number of digits is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The number of digits is: 2
```


Example – using a dataset

In this example, digits of the decimal format for each variable are returned from a dataset. It is known that the dataset contains four variables, but the digits of the decimal format for each variable are unknown. The result is written to a log.

```
DATA _NULL_;
  SET mydir.VARDFMAT (OBS=1);
  varstart = 'X';
  varcount = 4;

  DO i = 1 TO varcount;
    varname = CAT(varstart, i);
    varformat = VFORMATDX(varname);
    PUT 'The number of digits for ' varname' is ' varformat;
  END;

RUN;
```

This produces the following output:

```
The number of digits for X1 is 3
The number of digits for X2 is 0
The number of digits for X3 is 4
The number of digits for X4 is 2
```

The properties of the VARDFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8	F8.3		
X2	number	8	DATE9.		
X3	number	8	F8.4		
X4	number	8	F7.2		

VFORMATN

Returns the format name for a specified variable name.

➤ **VFORMATN** ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Var

The variable name.

Example

In this example, format names for specified variables are returned. `nums4` is the fourth variable in `ARRAY nums(5)`. Some of these variables have default formats, while *fraction* has an assigned format. The result is written to the `log`.

```
DATA _NULL_;  
  ARRAY nums(5) (10 22 36 4 54);  
  p2 = "Racing Colours";  
  FORMAT fraction F7.3;  
  fraction = 125.525;  
  
  var_nums4 = VFORMATN(nums4);  
  PUT "The format name for the nums4 variable is: " var_nums4;  
  
  var_p2 = VFORMATN(p2);  
  PUT "The format name for the p2 variable is: " var_p2;  
  
  var_frac = VFORMATN(fraction);  
  PUT "The format name for the fraction variable is: " var_frac;  
RUN;
```

This produces the following output:

```
The format name for the nums4 variable is: BEST  
The format name for the p2 variable is: $  
The format name for the fraction variable is: F
```

VFORMATNX

Returns the format name for a specified quoted variable name.

⇒ **VFORMATNX** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a format and value assigned to it. The format name for the specified variable is then returned. The result is written to a log.

```
DATA _NULL_;

  FORMAT sp BEST4.;
  sp = 3.4E+38;

  fmtv = VFORMATNX("sp");
  PUT "The format name is: " fmtv;

RUN;
```

This produces the following output:

```
The format name is: BEST
```

Example

In this example, a variable has a format and value assigned to it. This variable is then assigned to a second variable. The format name of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

  FORMAT sp BEST4.;
  sp = 3.4E+38;

  b = "sp";

  fmtv = VFORMATNX(b);
  PUT "The format name is: " fmtv;

RUN;
```

This produces the following output:

```
The format name is: BEST
```

Example – using a dataset

In this example, format names for variables are returned from a dataset. It is known that the dataset contains four variables, but the format names are unknown. The result is written to a log.

```
DATA _NULL_;
  SET mydir.VARFMAT (OBS=1);
  varstart = 'X';
  varcount = 4;

  DO i = 1 TO varcount;
    varname = CAT(varstart, i);
    varformat = VFORMATNX(varname);
    PUT 'The format name of ' varname ' is ' varformat;
  END;
RUN;
```

This produces the following output:

```
The format name of X1 is BEST
The format name of X2 is DATE
The format name of X3 is F
The format name of X4 is F
```

The properties of the VARFMT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8	BEST3.2		
X2	number	8	DATE9.		
X3	number	8	F8.4		
X4	number	8	F7.3		

VFORMATW

Returns the width of the format for the specified variable name.

```
⇒ VFORMATW ( ⇒ variable-name ) ⇒
```

Return type: Numeric

variable-name

Type: Var

The variable name.

Example

In this example, the widths of the formats for variables are returned. `nums(2)` is the second variable in `ARRAY nums(5)`. Some of these variables have default formats, while others have formats assigned. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (1 2 3 4 5);
  FORMAT p21 $10.;
  p21 = "First-Past-The-Post";
  sp = 3.4E+38;
  FORMAT date Date9.;
  date = "19-MAY-2017"D;

  var_nums2 = VFORMATW(nums(2));
  PUT "The width of the format for the nums(2) variable is: " var_nums2;

  var_p21 = VFORMATW(p21);
  PUT "The width of the format for the p21 variable is: " var_p21;

  var_sp = VFORMATW(sp);
  PUT "The width of the format for the sp variable is: " var_sp;

  var_date = VFORMATW(date);
  PUT "The width of the format for the date variable is: " var_date;
RUN;
```

This produces the following output:

```
The width of the format for the nums(2) variable is: 12
The width of the format for the p21 variable is: 10
The width of the format for the sp variable is: 12
The width of the format for the date variable is: 9
```

Example – using a dataset

In this example, variables are read from a dataset and their widths are returned. The result is written to the log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;

  SET mydir.BOOKS (OBS=1) ;

  ARRAY NB{*} _NUMERIC_ ;
  ARRAY CB{*} _CHARACTER_ ;

  hw = "has the width: ";
  sp = "          ";
  tv = "The variable: ";

  PUT tv;

  DO i = 1 TO DIM(NB);
    vn = VNAME(NB{i});
    vfw = VFORMATW(NB{i});
    PUT sp vn hw vfw;
  END;
```

```
DO i = 1 TO DIM(CB);  
    vn = VNAME(CB{i});  
    vfw = VFORMATW(CB{i});  
    PUT sp vn hw vfw;  
END;  
  
whw = VFORMATW(hw);  
  
PUT "    ";  
  
PUT "The variable whw has the width: " whw;  
RUN;
```

This produces the following output:

```
The variable:  
Purchased has the width: 9  
Dewey has the width: 9  
Price has the width: 9  
Title has the width: 114  
Author has the width: 23  
ISBN has the width: 14  
Subject has the width: 21  
  
The variable whw has the width: 15
```

The properties of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	\$14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

VFORMATWX

Returns the width of the format for the specified quoted variable name.

⇒ **VFORMATW X** ⇒ (⇒ ***variable-name*** ⇒) ⇒

Return type: Numeric

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a format and value assigned to it. The width of the format for the variable is then returned. The result is written to a log.

```
DATA _NULL_;

    FORMAT sp BEST4.;
    sp = 3.4E+38;

    fmtv = VFORMATWX("sp");
    PUT "The width of the format is: " fmtv;

RUN;
```

This produces the following output:

```
The width of the format is: 4
```

Example

In this example, a variable has a format and value assigned to it. This variable is then assigned to a second variable. The width of the format of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

    FORMAT sp BEST4.;
    sp = 3.4E+38;

    b = "sp";

    fmtv = VFORMATWX(b);
    PUT "The with of the format is: " fmtv;

RUN;
```

This produces the following output:

```
The width of the format is: 4
```

Example – using a dataset

In this example, widths of the formats for variables are returned from a dataset. It is known that the dataset contains four variables, but the widths of the formats for the variables are unknown. The result is written to a log.

```
DATA _NULL_;  
  SET mydir.VARWIDTH (OBS=1);  
  varstart = 'X';  
  varcount = 4;  
  
  DO i = 1 TO varcount;  
    varname = CAT(varstart, i);  
    varwidth = VFORMATWX(varname);  
    PUT 'The width of the format of ' varname ' is ' varwidth;  
  END;  
RUN;
```

This produces the following output:

```
The width of the format of X1 is 10  
The width of the format of X2 is 9  
The width of the format of X3 is 8  
The width of the format of X4 is 8
```

The properties of the VARWIDTH dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	character	10	\$10.		
X2	number	8	DATE9.		
X3	number	8	F8.4		
X4	number	8	F8.2		

VFORMATX

Returns the format for the specified quoted variable name.

⇒ **VFORMATX** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a format and value assigned to it. The format of the variable is then returned. The result is written to a log.

```
DATA _NULL_;

  FORMAT a F3.1;
  a = 2;

  fmtv = VFORMATX("a");
  PUT "The format is: " fmtv;

RUN;
```

This produces the following output:

```
The format is: F3.1
```

Example

In this example, a variable has a format and value assigned to it. This variable is then assigned to a second variable. The format of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

  FORMAT a F3.1;
  a = 2;
  b = "a";

  fmtv = VFORMATX(b);
  PUT "The format is: " fmtv;

RUN;
```

This produces the following output:

```
The format is: F3.1
```

Example – using a dataset

In this example, formats for variables are returned from a dataset. It is known that there are five variables in a dataset, but the formats for the variables are unknown. The result is written to a log.

```
DATA _NULL_;
  SET mydir.VARFMAT (OBS=1);
  varstart = 'X';
  varcount = 5;

  DO i = 1 TO varcount;
    varname = CAT(varstart, i);
    varfmt = VFORMATX(varname);
    PUT 'The value for ' varname 'is ' varfmt;
  END;

RUN;
```

This produces the following output:

```
The value for X1 is BEST3.2
The value for X2 is DATE9.
The value for X3 is F8.4
The value for X4 is F7.3
The value for X5 is $200.
```

The structure of the VARFMT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8	BEST3.2		
X2	number	8	Date9.		
X3	number	8	F8.4		
X4	number	8	F7.3		
X5	character	200	\$200.		

VINARRAY

Returns a value indicating whether the specified variable name is in an array.

⇒ **VINARRAY** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

1 if the variable name is in an array, 0 otherwise.

variable-name

Type: Var

The variable.

Example

In this example, the function is used to check whether the variable `nums(7)` is available. `nums(7)` is the seventh variable in `ARRAY nums(12)`. The result is written to the `log`.

```
DATA _NULL_;
  ARRAY nums(12);

  var = VINARRAY(nums(7));

  if var = 1 THEN PUT "The nums(7) variable is in an array: " var;
  ELSE PUT "The nums(7) variable is not part of an array: " var;

RUN;
```

This produces the following output:

```
The nums(7) variable is in an array: 1
```

VINARRAYX

Returns a value indicating whether the specified quoted variable name is in an array.

⇒ **VINARRAYX** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

1 if the variable with the specified name is in an array, 0 otherwise.

variable-name

Type: Character

The name of the variable.

Example

In this example, the function is used to indicate whether the variable `nums4` is available. `nums4` is the fourth variable name in `ARRAY nums(12)`. The result is written to the log.

```
DATA _NULL_;  
  ARRAY nums(12);  
  
  var = VINARRAYX("nums4");  
  
  if VAR = 1 THEN PUT "The nums4 variable is in an array: " var;  
  ELSE PUT "The nums4 variable is not part of an array: " var;  
  
RUN;
```

This produces the following output

```
The nums4 variable is in an array: 1
```

Example

In this example, the function is used to indicate whether the variable `nums8` is available. `nums8` is the eighth variable name in `ARRAY nums(12)`. This variable is then assigned to a second variable. The result is written to the log.

```
DATA _NULL_;  
  ARRAY nums(12);  
  
  abc = "nums8";  
  
  var = VINARRAYX(abc);  
  
  if VAR = 1 THEN PUT "The abc variable is in an array: " var;  
  ELSE PUT "The abc variable is not part of an array: " var;  
  
RUN;
```

This produces the following output

```
The abc variable is in an array: 1
```

VINFORMAT

Returns the informat for a specified variable name.

⇒ **VINFORMAT** (*variable-name*) ⇒

Return type: Character

variable-name

Type: Var

The variable name.

Example

In this example, informats for variables are returned. Some of these variables have default informats, while others have informats assigned. The result is written to the log.

```
DATA _NULL_;
  INFORMAT sp BEST4.;
  sp = 3.4E+38;
  paddle = 208.77 + RANNOR(153)*9.2076838;
  INFORMAT date Date9.;
  date = "12-MAY-2017"D;

  var_sp = VINFORMAT(sp);
  PUT "The informat for the sp variable is: " var_sp;

  var_paddle = VINFORMAT(paddle);
  PUT "The informat for the Paddle variable is: " var_paddle;

  var_date = VINFORMAT(date);
  PUT "The informat for the Date variable is: " var_date;
RUN;
```

This produces the following output:

```
The informat for the sp variable is: BEST4.
The informat for the paddle variable is: 32.
The informat for the date variable is: DATE9.
```

VINFORMATD

Returns the digits of the informat for the specified variable name.

⇒ **VINFORMATD** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

variable-name

Type: Var

The variable name.

Example – using a dataset

In this example, digits of the decimal informat for each variable are returned from a dataset. It is known that the dataset contains four named variables, but the digits of the informat for each variable are unknown. The result is written to the log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;

SET mydir.VARDFMAT (OBS=1);

var = VINFORMATD(i);
PUT "The number of digits for the i variable informat is: " var;

var = VINFORMATD(sp);
PUT "The number of digits for the sp variable informat is: " var;

var = VINFORMATD(dp);
PUT "The number of digits for the dp variable informat is: " var;

var = VINFORMATD(fraction);
PUT "The number of digits for the fraction variable informat is: " var;
RUN;
```

This produces the following output:

```
The number of digits for the i variable informat is: 2
The number of digits for the sp variable informat is: 3
The number of digits for the dp variable informat is: 4
The number of digits for the fraction variable informat is: 3
```

The properties of the VARDFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
dp	number	8		8.4	
fraction	number	8		7.3	
i	number	8		BEST3.2	
sp	number	8		4.3	

VINFORMATDX

Returns the digits of the informat for the specified quoted variable name.

⇒ VINFORMATDX (⇒ *variable-name*) ⇒

Return type: Numeric

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has an informat and value assigned to it. The number of digits for the informat of the variable are then returned. The result is written to a log.

```
DATA _NULL_;

  INFORMAT a F5.2;
  a = 2.369;

  fmtv = VINFORMATDX("a");
  PUT "The number of digits is: " fmtv;

RUN;
```

This produces the following output:

```
The number of digits is: 2
```

Example

In this example, a variable has an informat and value assigned to it. This variable is then assigned to a second variable. The number of digits for the informat of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

  INFORMAT a F5.2;
  a = 2.369;
  b = "a";

  fmtv = VINFORMATDX(b);
  PUT "The number of digits is: " fmtv;

RUN;
```

This produces the following output:

```
The number of digits is: 2
```

Example – using a dataset

In this example, digits of the decimal informat for each variable are returned from a dataset. It is known that the dataset contains four variables, but the digits of the decimal informat for each variable are unknown. The result is written to a log.

```
DATA _NULL_;  
  SET mydir.VARDFMAT (OBS=1);  
  varstart = 'X';  
  varcount = 4;  
  
  DO i = 1 TO varcount;  
    varname = CAT(varstart, i);  
    vardigit = VINFORMATDX(varname);  
    PUT 'The number of digits for ' varname'is ' vardigit;  
  END;  
  
RUN;
```

This produces the following output:

```
The number of digits for X1 is 2  
The number of digits for X2 is 3  
The number of digits for X3 is 4  
The number of digits for X4 is 3
```

The properties of the VARDFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8		BEST3.2	
X2	number	8		4.3	
X3	number	8		8.4	
X4	number	8		7.3	

VINFORMATN

Returns the informat name for a specified variable name.

➤ **VINFORMATN** ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Var

The variable name.

In this example, the informat names for specified variables are returned. The `p21` variable uses a default informat, while the other variables have informats assigned. The result is written to the log.

```
DATA _NULL_;
  INFORMAT i BEST2.;
  i = 10;
  p21 = "Twenty-One-Today";
  INFORMAT date Date9.;
  date = "19-MAY-2017"D;

  var = VINFORMATN(i);
  PUT "The informat name for the i variable is: " var;

  var = VINFORMATN(p21);
  PUT "The informat name for the p21 variable is: " var;.

  var = VINFORMATN(date);
  PUT "The informat name for the date variable is: " var;
RUN;
```

This produces the following output:

```
The informat name for the i variable is: BEST
The informat name for the p21 variable is: $
The informat name for the date variable is: DATE
```

VINFORMATNX

Returns the informat name for a specified quoted variable name.

➤ **VINFORMATNX** ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, an informat name for a specified variable is returned. The result is written to a log.

```
DATA _NULL_;

    INFORMAT a F3.1;
    a = 2;

    fmtv = VINFORMATNX("a");
    PUT "The informat name is: " fmtv;

RUN;
```

This produces the following output:

```
The informat name is: F
```

Example

In this example, a variable has an informat and value assigned to it. This variable is then assigned to a second variable. The informat name of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

    INFORMAT a F3.1;
    a = 2;
    b = "a";

    fmtv = VINFORMATNX(b);
    PUT "The informat name is: " fmtv;

RUN;
```

This produces the following output:

```
The informat name is: F
```

Example – using a dataset

In this example, informat names for variables are returned from a dataset. It is known that the dataset contains four variables, but the informat names are unknown. The result is written to a log.

```
DATA _NULL_;
    SET mydir.VARNFMT (OBS=1);
    varstart = 'X';
    varcount = 4;

    DO i = 1 TO varCount;
        varname = CAT(varstart, i);
        varformat = VINFORMATNX(varName);
        PUT 'The informat name for ' varname ' is ' varformat;
    END;

RUN;
```

This produces the following output:

```
The informat name for X1 is BEST
The informat name for X2 is DATE
The informat name for X3 is F
The informat name for X4 is F
```

The properties of the VARNFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8		BEST3.2	
X2	number	8		DATE9.	
X3	number	8		F8.4	
X4	number	8		F7.3	

VINFORMATW

Returns the width of the informat for the specified variable name.

```
⇒ VINFORMATW ( variable-name ) ⇐
```

Return type: Numeric

variable-name

Type: Var

The variable name.

Example

In this example, widths of the informats for the variables are returned. `nums(3)` is the third variable in `ARRAY nums(5)`. Some of these variables have default informats, while others have informats assigned. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (11 243 386 493 5);
  INFORMAT P21 $10.;
  p21 = "First-Past-The-Post";
  INFORMAT paddle $8.;
  paddle = 208.77 + RANNOR(153)*9.2076838;
  INFORMAT date Date9.;
  date = "19-MAY-2017"D;

  var = VINFORMATW(nums(3));
  PUT "The width of the informat for the nums(3) variable is: " var;

  var = VINFORMATW(p21);
  PUT "The width of the informat for the p21 variable is: " var;

  VAR = VINFORMATW(paddle);
  PUT "The width of the informat for the paddle variable is: " var;
```

```
var = VINFORMATW(date);  
PUT "The width of the informat for the date variable is: " var;  
RUN;
```

This produces the following output:

```
The width of the informat for the nums(3) variable is: 32  
The width of the informat for the p21 variable is: 10  
The width of the informat for the paddle variable is: 8  
The width of the informat for the date variable is: 9
```

VINFORMATWX

Returns the width of the informat for the specified quoted variable name.

⇒ **VINFORMATW X** ⇒ (⇒ *variable-name* ⇒) ⇒

Return type: Numeric

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has an informat and value assigned to it. The width of the informat for the variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  
INFORMAT sp BEST4.;  
sp = 3.4E+38;  
  
fmtv = VINFORMATWX("sp");  
PUT "The width of the informat is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The width of the informat is: 4
```

Example

In this example, a variable has an informat and value assigned to it. This variable is then assigned to a second variable. The width of the informat of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

    INFORMAT sp BEST4.;
    sp = 3.4E+38;

    b = "sp";

    fmtv = VINFORMATWX(b);
    PUT "The with of the informat is: " fmtv;

RUN;
```

This produces the following output:

```
The width of the informat is: 4
```

Example – using a dataset

In this example, widths of informats for variables are returned from a dataset. It is known that the dataset contains four variables, but the widths of the informats are unknown. The result is written to a log.

```
DATA _NULL_;
    SET mydir.VARIWIDTH (OBS=1);
    varstart = 'X';
    varcount = 4;

    DO i = 1 TO varcount;
        varname = CAT(varstart, i);
        varformat = VINFORMATWX(varName);
        PUT 'The width of the informat of ' varname ' is ' varformat;
    END;

RUN;
```

This produces the following output:

```
The width of the informat of X1 is 10
The width of the informat of X2 is 9
The width of the informat of X3 is 8
The width of the informat of X4 is 8
```

The properties of the VARIWIDTH dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	character	10		\$10.	
X2	number	8		DATE9.	
X3	number	8		F8.4	
X4	number	8		F8.2	

VINFORMATX

Returns the informat for a specified quoted variable name.

➤ VINFORMATX ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has an informat and value assigned to it. The informat for the variable is returned. The result is written to a log.

```
DATA _NULL_;

    INFORMAT sp BEST4.;
    sp = 3.4E+38;

    fmtv = VINFORMATX("sp");
    PUT "The informat is: " fmtv;

RUN;
```

This produces the following output:

```
The informat is: BEST4
```

Example

In this example, a variable has an informat and value assigned to it. This variable is then assigned to a second variable. The informat of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

    INFORMAT sp BEST4.;
    sp = 3.4E+38;

    b = "sp";

    fmtv = VINFORMATX(b);
    PUT "The informat is: " fmtv;

RUN;
```

This produces the following output:

```
The informat is: BEST4
```

Example – using a dataset

In this example, informats for variables are returned from a dataset. It is known that the dataset contains five variables, but the informats are unknown. The result is written to a log.

```
DATA _NULL_;  
  SET mydir.VARIFMAT (OBS=1);  
  varstart = 'X';  
  varcount = 5;  
  
  DO i = 1 TO varcount;  
    varname = CAT(varstart, i);  
    varformat = VINFORMATX(varname);  
    PUT 'The informat of ' varname ' is ' varformat;  
  END;  
RUN;
```

This produces the following output:

```
The informat of X1 is $14.  
The informat of X2 is DATE9.  
The informat of X3 is F8.4  
The informat of X4 is F7.3  
The informat of X5 is $200.
```

The properties of the VARIFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	character	14		\$14.	
X2	number	8		DATE9.	
X3	number	8		F8.4	
X4	number	8		F7.3	
X5	character	200		\$200.	

VLABEL

Returns the label of the specified variable name.

➤ **VLABEL** ➤ (➤ ***variable-name*** ➤) ➤

Return type: Character

variable-name

Type: Var

The variable name.

Basic Example

In this example variable labels are returned. `nums3` is the third variable in `ARRAY nums(5)`. Where the variable is not assigned a label, the default value is returned. The default label for a variable is its name. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (1 2 3 4 5);
  i = "ABCDEFGH";
  s1 = 25500;
  s2 = 5000;
  s3 = s1 - s2;
  LABEL s1 = 'Gross Income: ';
  LABEL s2 = 'Expenses: ';

  var = VLABEL(nums3);
  PUT "The nums3 variable has the following label: " var;

  var = VLABEL(i);
  PUT "The i variable has the following label: " var;

  var = VLABEL(s1);
  PUT "The s1 variable has the following label: " var;

  var = VLABEL(s2);
  PUT "The s2 variable has the following label: " var;

  var = s3;
  PUT "The difference between s1 and s2 is: " var;
RUN;
```

This produces the following output:

```
The nums3 variable has the following label: nums3
The i variable has the following label: i
The s1 variable has the following label: Gross Income:
The s2 variable has the following label: Expenses:
The difference between s1 and s2 is: 20500
```

Example – using a dataset

In this example, variable labels are returned from a dataset. The results are written to the log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;

  SET mydir.BOOKS (OBS=1);
  ARRAY BKNUM{*} _NUMERIC_ ;
  ARRAY BKCHAR{*} _CHARACTER_ ;

  DO i = 1 TO DIM(BKNUM);
    v1 = VLABEL(BKNUM{i});
    PUT "Variable " i "has the label: " v1;

  END ;

  DO j = 1 TO DIM(BKCHAR);
    v1 = VLABEL(BKCHAR{j});
    PUT "Variable " i "has the label: " v1;

    i = i + 1;

  END;

RUN;
```

This produces the following output:

```
Variable 1 has the label: Date book purchased
Variable 2 has the label: Dewey decimal number
Variable 3 has the label: Original price paid for book
Variable 4 has the label: Book title
Variable 5 has the label: Author of the book
Variable 6 has the label: International standard book number
Variable 7 has the label: BIC subject class
```

The structure of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	\$14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

VLABELX

Returns the label of the specified quoted variable name.

```
VLABELX ( variable-name )
```


Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable label is returned. The result is written to a log.

```
DATA _NULL_;

    s1 = 75000;
    LABEL s1 = 'Gross Income: ';

    fmtv = VLABELX("s1");
    PUT "The variable has the following label: " fmtv;

RUN;
```

This produces the following output:

```
The variable has the following label: Gross Income:
```

Example

In this example, the variable is then assigned to a second variable. The label for the first variable is then returned. The result is written to a log.

```
DATA _NULL_;

    s1 = 75000;
    LABEL s1 = 'Gross Income: ';

    b = "s1";

    fmtv = VLABELX(b);
    PUT "The variable has the following label: " fmtv;

RUN;
```

This produces the following output:

```
The variable has the following label: Gross Income:
```

Example – using a dataset

In this example, labels for variables are returned from a dataset. It is known that a dataset contains seven variables, but the labels for the variables are unknown. The result is written to a log.

```
DATA _NULL_;
  SET mydir.BOOKSX (OBS=1);
  varstart = 'X';
  varcount = 7;

  DO i = 1 TO varcount;
    varname = CAT(varstart, i);
    varformat = VLABELX(varname);
    PUT 'The variable ' varname' has the following label: ' varformat;
  END;
RUN;
```

This produces the following output:

```
The variable X1  has the following label: Book title
The variable X2  has the following label: Author of the book
The variable X3  has the following label: BIC subject class
The variable X4  has the following label: International standard book number
The variable X5  has the following label: Date book purchased
The variable X6  has the following label: Dewey decimal number
The variable X7  has the following label: Original price paid for book
```

The properties of the BOOKSX dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	character	114	\$114.	\$114.	Book title
X2	character	23	\$23.	\$23.	Author of the book
X3	character	21	\$21.	\$21.	BIC subject class
X4	character	14	\$14.	\$14.	International standard book number
X5	number	8	Date9.	Date9.	Date book purchased
X6	number	8	BEST9.	BEST9.	Dewey decimal number
X7	number	8	BEST9.	BEST9.	Original price paid for book

VLENGTH

Returns the length of the specified variable name.

```
➤ VLENGTH ➤ ( ➤ variable-name ➤ ) ➤
```

Return type: Numeric

The variable length.

variable-name

Type: Var

The variable name.

In this example, variable lengths in memory are returned. The result is written to the log.

Note:

Ensure the `ENCODING` is set to UTF-8 in both the file and local server to handle the string variable in Japanese.

```
DATA _NULL_;
  FORMAT sp BEST3.;
  sp = 3.4E+38;
  FORMAT date Date6.;
  date = "19-MAY-2017"D;
  FORMAT p21 $10.;
  p21 = "Twenty-One-Today";
  FORMAT p22 $50.;
  p22 = "ときがら茶:とうきん煤竹 空五倍子色";

  var = VLENGTH(sp);
  PUT "The length for the sp variable is: " var;

  PUT sp;

  var = VLENGTH(date);
  PUT "The length for the date variable is: " var;

  PUT date;

  var = VLENGTH(p21);
  PUT "The length for the p21 variable is: " var;

  PUT p21;

  var = VLENGTH(p22);
  PUT "The length for the p22 variable is: " var;

  PUT p22;
RUN;
```

This produces the following output:

```
The length for the sp variable is: 8
***
The length for the date variable is: 8
19MAY
The length for the p21 variable is: 10
Twenty-One
The length for the p22 variable is: 50
ときがら茶:とうきん煤竹 空五倍子色
```

In this example, the system recognised that it required at least eight bits to process the `sp` and `date` numeric variables. However, the system did not have sufficient space to display the `sp` variable, leaving asterisks in place. Similarly, only four characters of the `date` variable could be displayed.

While the `p21` variable can be handled correctly in the log output, the `p22` Japanese string variable cannot. This is because each Japanese character can be a number of bytes. Any specified string length falling short on a character in the character string would result in only the length of the character string being displayed. The character string output is therefore unknown. For example, if the `p22` variable was set to `$44.`, this would display: `ときから茶:とうきん爆竹 空五倍`, which is two characters short of the full string. However, if the `p22` variable was set to `$48.` which falls short on a character, then only the length of the character string would be displayed.

VLENGTHX

Returns the length of the specified quoted variable `name`.

⇒ **VLENGTHX** (*variable-name*) ⇒

Return type: Numeric

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a value assigned to it. The length of the variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  sp = 3.4E+38;  
  vrlgth = VLENGTHX("sp");  
  PUT "The length of the variable is: " vrlgth;  
RUN;
```

This produces the following output:

```
The length of the variable is: 8
```

Example

In this example, a variable has a value assigned to it. This variable is then assigned to a second variable. The length of first variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  sp = 3.4E+38;  
  b = "sp";  
  vrlgth = VLENGTHX(b);  
  PUT "The length of the variable is: " vrlgth;  
RUN;
```

This produces the following output:

```
The length of the variable is: 8
```

Example – using a dataset

In this example, lengths for variables are returned from a dataset. It is known that the dataset contains four variables, but the lengths for the variables are unknown. The result is written to a log.

```
DATA _NULL_;
  SET mydir.VARLGTH (OBS=1);
  varstart = 'X';
  varcount = 4;

  DO i = 1 TO varCount;
    varName = CAT(varstart, i);
    varFormat = VLENGTHX(varName);
    PUT 'The length for the ' varname 'variable is ' varformat;
  END;
RUN;
```

This produces the following output:

```
The length for the X1 variable is 8
The length for the X2 variable is 8
The length for the X3 variable is 10
The length for the X4 variable is 50
```

The properties of the VARWIDTH dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8	BEST\$10.		
X2	number	8	DATE9.		
X3	character	10	\$10.		
X4	character	50	\$50.		

VNAME

Returns the name of a specified variable *name*.

```
⇒ VNAME ( variable-name ) ⇒
```

Return type: Character

variable-name

Type: Var

The variable name.

Example – using a dataset

In this example, variable names are returned from a dataset.

```
LIBNAME mydir "C:\temp";
DATA _NULL_;
  SET mydir.BOOKS (OBS=1);
  ARRAY BKNUM{*} _NUMERIC_ ;
  ARRAY BKCHAR{*} _CHARACTER_ ;
  length cn $200. ;

  DO i = 1 TO DIM(BKNUM);
    vn = VNAME(BKNUM{i});
    cn = CATX(" ", cn, vn);
  END ;

  DO i = 1 TO DIM(BKCHAR);
    vn = VNAME(BKCHAR{i});
    cn = CATX(" ", cn, vn);
  END;

  PUT cn;
RUN;
```

This produces the following output:

```
Purchased, Dewey, Price, Title, Author, ISBN, Subject
```

The properties of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	S14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

VNAMEX

Returns the name of a specified quoted variable name.

➤ **VNAMEX** ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, the variable has a value assigned to it. The name of the variable is returned. The result is written to a log.

```
DATA _NULL_;  
  a = 2;  
  fmtv = VNAMEX("a");  
  PUT "The name of the variable is: " fmtv;  
RUN;
```

This produces the following output:

```
The name of the variable is: a
```

Example

In this example, a variable has a value assigned to it. This variable is then assigned to a second variable. The name of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  a = 2;  
  b = "a";  
  fmtv = VNAMEX(b);  
  PUT "The name of the variable is: " fmtv;  
RUN;
```

This produces the following output:

```
The name of the variable is: a
```

Example

In this example, the variable names are returned. `nums4` is the fourth variable name in `ARRAY nums(5)`. The result is written to the log.

```
DATA _NULL_;  
  ARRAY nums(5) (10 20 33 401 5);  
  p2 = "1234567";  
  fraction = 125.525;  
  
  var_a4 = "nums4";  
  
  var_n4 = VNAMEX("nums4");  
  PUT "The name for the nums4 variable is: " var_n4;  
  
  var4 = VNAMEX(var_a4);  
  PUT "The name for the var_a4 variable is: " var4;  
  
  var_p2 = VNAMEX("p2");  
  PUT "The name for the p2 variable is: " var_p2;  
  
  var_frac = VNAMEX("fraction");  
  PUT "The name for the fraction variable is: " var_frac;  
RUN;
```

This produces the following output:

```
The name for the nums4 variable is: nums4
The name for the var_a4 variable is: nums4
The name for the p2 variable is: p2
The name for the fraction variable is: fraction
```

VTRANSCODE

Returns a value indicating whether a specified variable name can be transcoded.

➤ **VTRANSCODE** ➤ (➤ *variable-name* ➤) ➤

Return type: Numeric

1 if the variable is transcoded, or 0 otherwise. The default VTRANSCODE setting is set to 1.

variable-name

Type: Var

The variable name.

Example

In this example, it has been decided that variable p21 is not to be transcoded. An attribute statement ATTRIB p21 TRANSCODE=NO is added for variable p21. The variable for p22 is to be transcoded, so this time the added attribute statement becomes ATTRIB p22 TRANSCODE=YES. The result is written to the log.

```
DATA _NULL_;
  FORMAT p21 $10.;
  p21 = "Twenty-One-Today";
  ATTRIB p21 TRANSCODE=NO;
  p22 = "BIRTHDAY GIRL";
  ATTRIB p22 TRANSCODE=YES;
  i = 100;

  var = VTRANSCODE(p21);

  If var = 1 THEN PUT "The variable p21 will be transcoded: " var;
  ELSE PUT "The variable p21 will not be transcoded: " var;

  var = VTRANSCODE(p22);

  If var = 1 THEN PUT "The variable p22 will be transcoded: " var;
  ELSE PUT "The variable p22 will not be transcoded: " var;

  VAR = VTRANSCODE(i);

  If var = 1 THEN PUT "The variable i will be transcoded: " var;
  ELSE PUT "The variable i will not be transcoded: " var;
```



```
RUN;
```

This produces the following output:

```
The variable p21 will not be transcoded: 0
The variable p22 will be transcoded: 1
The variable i will be transcoded: 1
```

It can be seen from this example that both variables `p22` and `i` will be transcoded.

VTRANSCODEX

Returns a value indicating whether a specified quoted variable name can be transcoded.

➤ **VTRANSCODEX** ➤ (➤ *variable-name* ➤) ➤

Return type: Numeric

1 if the variable is transcoded, or 0 otherwise. The default `VTRANSCODE` setting is set to 1.

variable-name

Type: Character

The variable name in quotes.

Example

In this example, it has been decided that variable `p21` is not to be transcoded. An attribute statement `ATTRIB p21 TRANSCODE=NO` is added for variable `p21`. The variable for `p22` is to be transcoded, so this time the added attribute statement becomes `ATTRIB p22 TRANSCODE=YES`. The result is written to the log.

```
DATA _NULL_;
  FORMAT p21 $10.;
  p21 = "Twenty-One-Today";
  ATTRIB p21 TRANSCODE=NO;
  p22 = "BIRTHDAY BOY";
  ATTRIB p22 TRANSCODE=YES;
  i = 100;

  var = VTRANSCODEX("p21");

  If var = 1 THEN PUT "The variable p21 will be transcoded: " var;
  ELSE PUT "The variable p21 will not be transcoded: " var;

  var = VTRANSCODEX("p22");

  If VAR = 1 THEN PUT "The variable p22 will be transcoded: " var;
  ELSE PUT "The variable p22 will not be transcoded: " var;
```

```
var = VTRANSCODEX("i");
```

```
If var = 1 THEN PUT "The variable i will be transcoded: " var;  
ELSE PUT "The variable i will not be transcoded: " var;
```

```
RUN;
```

This produces the following output:

```
The variable p21 will not be transcoded: 0  
The variable p22 will be transcoded: 1  
The variable i will be transcoded: 1
```

It can be seen from this example that both variables `p22` and `i` will be transcoded.

VTYPE

Returns the type of data associated with the specified variable name. This will be `N` for numeric or `C` for character.

```
⇒ VTYPE ⇒ ( ⇒ variable-name ⇒ ) ⇒
```

Return type: Character

variable-name

Type: Var

The variable name.

Basic Example

In this example, several variables are checked with the function. `nums(5)` is the fifth variable in `ARRAY nums(5)`. The result is written to the log.

```
DATA _NULL_;  
  ARRAY nums(5) (11 29 304 405 55);  
  p21 = "Twenty-One";  
  dp = 1.7E+308;  
  dollar = 125;  
  date = "27-MAY-1954"D;  
  
  var = VTYPE(nums(5));  
  PUT "The variable type for the nums(5) variable is: " var;  
  
  var = VTYPE(p21);  
  PUT "The variable type for the p21 variable is: " var;  
  
  var = VTYPE(dp);  
  PUT "The variable type for the dp variable is: " var;  
  
  var = VTYPE(dollar);
```

```
PUT "The variable type for the dollar variable is: " var;  
  
var = VTYPE(date);  
PUT "The variable type for the date variable is: " var;  
RUN;
```

This produces the following outputs:

```
The variable type for the nums(5) variable is: N  
The variable type for the p21 variable is: C  
The variable type for the dp variable is: N  
The variable type for the dollar variable is: N  
The variable type for the date variable is: N
```

Example – using a dataset

In this example, all the variable names, and variable numerical and character types are returned from a dataset.

```
LIBNAME mydir "c:\temp";  
DATA _NULL_;  
    SET mydir.BOOKS (OBS=1) ;  
  
    ARRAY BKNUM{*} _NUMERIC_ ;  
    ARRAY BKCHAR{*} _CHARACTER_ ;  
  
    DO i = 1 TO DIM(BKNUM);  
        vn = VNAME(BKNUM{i});  
        vl = VTYPE(BKNUM{i});  
        PUT "The variable " vn "has type: " vl;  
    END ;  
  
    DO i = 1 TO DIM(BKCHAR);  
        vn = VNAME(BKCHAR{i});  
        vl = VTYPE(BKCHAR{i});  
        PUT "The variable " vn "has type: " vl;  
    END;  
RUN;
```

This produces the following outputs:

```
The variable Purchased has type: N  
The variable Dewey has type: N  
The variable Price has type: N  
The variable Title has type: C  
The variable Author type: C  
The variable ISBN type: C  
The variable Subject has type: C
```

The properties of the BOOKS dataset:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	S14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

VTYPEX

Returns the type of data associated with the specified quoted variable name. This will be **N** for numeric or **C** for character.

```
⇒ VTYPEX ( variable-name ) ⇒
```

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a value assigned to it. The type of data associated with the variable is then returned. The result is written to a log.

```
DATA _NULL_;

FORMAT a F3.1;
a = 2;

fmtv = VTYPEX("a");
PUT "The variable type is: " fmtv;

RUN;
```

This produces the following output:

```
The variable type is: N
```

Example

In this example, a variable has a value assigned to it. This variable is then assigned to a second variable. The type of data associated with the first variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  FORMAT a F3.1;  
  a = 2;  
  b = "a";  
  
  fmtv = VTYPEX(b);  
  PUT "The variable type is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The variable type is: N
```

Example – using a dataset

In this example, the type of data associated with each variable is returned from a dataset. It is known that the dataset contains seven variables, but the type of data associated with each variable is unknown. The result is written to a log.

```
DATA _NULL_;  
  SET mydir.BOOKSX (OBS=1);  
  varstart = 'X';  
  varcount = 7;  
  
  DO i = 1 TO varcount;  
    varname = CAT(varstart, i);  
    vartype = VTYPEX(varname);  
    PUT 'The variable ' varname 'is ' vartype;  
  END;  
RUN;
```

This produces the following output:

```
The variable X1 is C  
The variable X2 is C  
The variable X3 is C  
The variable X4 is C  
The variable X5 is N  
The variable X6 is N  
The variable X7 is N
```

The properties of the BOOKSX dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	character	114	\$114.	\$114.	Book title
X2	character	23	\$23.	\$23.	Author of the book
X3	character	21	\$21.	\$21.	BIC subject class
X4	character	14	\$14.	\$14.	International standard book number
X5	number	8	Date9.	Date9.	Date book purchased
X6	number	8	BEST9.	BEST9.	Dewey decimal number
X7	number	8	BEST9.	BEST9.	Original price paid for book

VVALUE

Returns the formatted value of the specified variable name.

```
➤ VVALUE ( variable-name ) ➤
```

Return type: Character

variable-name

Type: Var

The variable name.

Example

In this example, the formatted values of the variables are returned. `nums(3)` is the third variable in `ARRAY nums(5)`. The result is written to the log.

```
DATA _NULL_;
  ARRAY nums(5) (12 27 37 40 5);
  p2 = "Epsom Derby";
  sp = 3.4E+38;
  dp1 = 1.7976931348623158E+308;
  paddle = 208.77 + RANNOR(153)*9.2076838 + 5;
  FORMAT date Date9.;
  date = "14-JUN-2017"D;

  var = VVALUE(nums(3));
  PUT "The value for the nums(3) variable is: " var;

  var = VVALUE(p2);
  PUT "The value for the p2 variable is: " var;

  var = VVALUE(dp1);
  PUT "The value for the dp1 variable is: " var;

  var = VVALUE(paddle);
  PUT "The value for the paddle variable is: " var;

  var = VVALUE(date);
```

```
PUT "The value for the date variable is: " var;  
RUN;
```

This produces the following output:

```
The value for the nums(3) variable is: 3  
The value for the p2 variable is: Epsom Derby  
The value for the dp1 variable is: 1.797693E308  
The value for the paddle variable is: 201.08512308  
The value for the date variable is: 14JUN2017
```

In this example strings are retained, while numerical variables can be processed.

VVALUEX

Returns the formatted value of the specified quoted variable name.

➤ **VVALUEX** ➤ (➤ *variable-name* ➤) ➤

Return type: Character

variable-name

Type: Character

The quoted variable name.

Example

In this example, a variable has a value assigned to it. The formatted value of a variable is returned. The result is written to a log.

```
DATA _NULL_;  
FORMAT a F3.1;  
a = 2;  
  
fmtv = VVALUEX("a");  
PUT "The value of the variable is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The value of the variable is: 2.0
```

Example

In this example, a variable has a value assigned to it. This variable is then assigned to a second variable. The formatted value of the first variable is then returned. The result is written to a log.

```
DATA _NULL_;  
  FORMAT a F3.1;  
  a = 2;  
  b = "a";  
  
  fmtv = VVALUEX(b);  
  PUT "The value of the variable is: " fmtv;  
  
RUN;
```

This produces the following output:

```
The value of the variable is: 2.0
```

Example – using a dataset

In this example, the formatted value for each variable is returned from a dataset. It is known that the dataset contains four variables, but the formatted value for each variable is unknown. The result is written to a log.

```
DATA _NULL_;  
  SET mydir.VARNFMAT (OBS=1);  
  varstart = 'X';  
  varcount = 4;  
  
  DO i = 1 TO varcount;  
    varname = CAT(varstart, i);  
    varvalue = VVALUEX(varName);  
    PUT 'The value for ' varname 'is ' varvalue;  
  END;  
  
RUN;
```

This produces the following output:

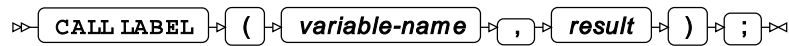
```
The value for X1 is 10  
The value for X2 is 21216  
The value for X3 is 1.7E308  
The value for X4 is 125.525
```

The properties of the VARNFMAT dataset is as follows:

Name	Type	Length	Format	Informat	Label
X1	number	8	BEST3.2		
X2	number	8	DATE9.		
X3	number	8	F8.4		
X4	number	8	F7.3		

CALL LABEL

Returns the label for a specified variable name. The label is returned as an argument in the `CALL` routine.



variable-name

Type: Var

The variable name for which to retrieve the label.

result

Type: Character

A character value that will be assigned the label of the first variable.

Example – using a dataset

In this example, the labels are copied from a dataset. The result is written to a log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;
  SET mydir.BOOKS (OBS=1) ;
  ARRAY BKNUM{*} _NUMERIC_ ;
  ARRAY BKCHAR{*} _CHARACTER_ ;

  FORMAT label $35.;

  DO i = 1 TO DIM(BKNUM);
    CALL LABEL(BKNUM{i}, label);
    PUT "Variable " i "has the label: " label;
  END;

  DO j = 1 TO DIM(BKCHAR);
    CALL LABEL(BKCHAR{j}, label);
    PUT "Variable " i "has the label: " label;
    i = i + 1;
  END;

RUN;
```

This produces the following output:

```
Variable 1 has the label: Date book purchased
Variable 2 has the label: Dewey decimal number
Variable 3 has the label: Original price paid for book
Variable 4 has the label: Book title
Variable 5 has the label: Author of the book
Variable 6 has the label: International standard book number
Variable 7 has the label: BIC subject class
```

The structure of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	S14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

CALL VNAME

Returns the name for a specified variable name. The name is returned as an argument in the CALL routine.

```
CALL VNAME ( variable-name , result ) ;
```

variable-name

Type: Var

The variable for which to retrieve the name.

result

Type: Character

A character value that will be assigned the name of the first variable.

Example – using a dataset

In this example, the variable names are copied from a dataset. The result is written to a log.

```
LIBNAME mydir "c:\temp";
DATA _NULL_;

  SET mydir.BOOKS (OBS=1) ;
  ARRAY BKNUM{*} _NUMERIC_ ;
  ARRAY BKCHAR{*} _CHARACTER_ ;

  FORMAT name $35.;

  DO i = 1 TO DIM(BKNUM);

    CALL VNAME(BKNUM{i}, name);
    PUT "Variable " i "has the name: " name;

  END;

  DO j = 1 TO DIM(BKCHAR);

    CALL VNAME(BKCHAR{j}, name);
    PUT "Variable " i "has the name: " name;

    i = i + 1;

  END;

RUN;
```

This produces the following output:

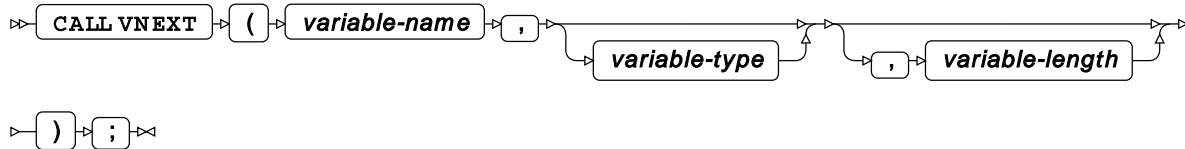
```
Variable 1 has the name: Purchased
Variable 2 has the name: Dewey
Variable 3 has the name: Price
Variable 4 has the name: Title
Variable 5 has the name: Author
Variable 6 has the name: ISBN
Variable 7 has the name: Subject
```

The properties of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	S14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

CALL VNEXT

Returns the variable names from a `DATA` step.



When the routine is first called, the first variable in the `DATA` step's variable list is returned to `varname`. For each subsequent call, the subsequent variable name is returned. It can also supply the variable type and variable length.

variable-name

Type: Character

The name of the variable.

variable-type

Optional argument

Type: Character

The variable to provide the type of data associated with it (the type of data will be `N` for numeric or, `C` for a character).

variable-length

Optional argument

Type: Numeric

The variable to provide the length.

Example – using a dataset

In this example, all the variables in the `DATA` step are returned. The result is written to a log.

```

LIBNAME mydir "C:\temp";
DATA _NULL_;

  SET mydir.BOOKS (OBS=1);

  FORMAT varnext $20.;
  vtype="";
  vlength=0;

  DO UNTIL (varnext = "");
    CALL VNEXT (varnext,vtype, vlength);
    IF varnext != "" THEN PUT "Variable: " Varnext "Type: " vtype "Length: "
  vlength;
  END;
RUN;
  
```

This produces the following output:

```
Variable: _N_ Type: N Length: 8
Variable: _ERROR_ Type: N Length: 8
Variable: _IORC_ Type: N Length: 8
Variable: Title Type: C Length: 114
Variable: Author Type: C Length: 23
Variable: ISBN Type: C Length: 14
Variable: Subject Type: C Length: 21
Variable: Purchased Type: N Length: 8
Variable: Dewey Type: N Length: 8
Variable: Price Type: N Length: 8
Variable: varnext Type: C Length: 20
Variable: vtype Type: C Length: 1
Variable: vlength Type: N Length: 8
```

The properties of the BOOKS dataset is as follows:

Name	Type	Length	Format	Informat	Label
Author	character	23	\$23.	\$23.	Author of the book
Dewey	number	8	BEST9.		Dewey decimal number
ISBN	character	14	S14.	\$14.	International standard book number
Price	number	8	BEST9.		Original price paid for book
Purchased	number	8	DATE9.	DATE9.	Date book purchased
Subject	character	21	\$21.	\$21.	BIC subject class
Title	character	114	\$114.	\$114.	Book title

Web functions

Convert the text in URLs and HTML files to different forms.

Hypertext Markup Language (HTML) uses a language that contains certain reserved characters. A *Uniform Resource Locator* (URL) uses a limited set of characters. If an HTML file contains characters in the reserved set of characters, they are converted to entities; if a URL contains characters outside the available set, the characters are replaced by escaped characters. The functions in this group enable you to perform these conversions and replacements.

HTMLDECODE ↗	2216
Converts an HTML entity in a string to its equivalent session encoding character.	
HTMLENCODE ↗	2217
Converts a character in a string to its equivalent HTML character entity for the session encoding.	
URLDECODE ↗	2219
Returns the result of replacing escaped characters in a URL with the equivalent session character.	
URLENCODE ↗	2220
Converts a non-ASCII or special character used in a URL to a code.	

HTMLDECODE

Converts an HTML entity in a string to its equivalent session encoding character.

» **HTMLDECODE** « (*expression*) «»

Hypertext Markup Language (HTML) contains various characters that are reserved in the language, such as quotation marks (" and ') and angle brackets (< and >). However, a Web page might need to display these characters. To enable this, the characters are replaced with *character entities*, which are codes that represent characters. For example, if a Web page contained the text:

```
value < 1 and > 7
```

the angle brackets would be interpreted as the delimiters for HTML tags, and the text would display incorrectly. To ensure the text displayed correctly the brackets can be replaced by named character entities (in this case, `<` and `>`).

Other characters can also be represented using entities; this is particularly useful for characters outside the 7-bit ASCII character set that you might want to represent in different session encodings. For example, the character ä can be represented by the entity `ä` .

Characters outside the 7-bit ASCII character set are encoded using a decimal entity.

The following named entities can be decoded:

<code>&gt;</code>	>
<code>&lt;</code>	<
<code>&amp;</code>	&
<code>&apos;</code>	'
<code>&quot;</code>	"

Decimal numeric entities can also be decoded. For example, if a string to be decoded contains `Ö` , the character Ö is returned.

Return type: Character

expression

Type: Character

A string of one or more characters that contains character entities.

Basic example

In this example, the function converts character entities to their character equivalents. The result is written to the log.

```
DATA _NULL_;  
    result = HTMLDECODE("value &lt; 1 and &gt; 7");  
    PUT result;  
RUN;
```

This produces the following output:

```
value < 1 and > 7
```

Example – decoding decimal numeric entity

In this example, the function converts decimal numeric entities to their character equivalents. The result is written to the log.

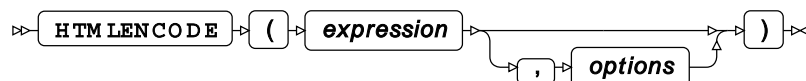
```
DATA _NULL_;  
    result = HTMLDECODE("Die B&#228;ren Caf&#233;");  
    PUT result;  
RUN;
```

This produces the following output:

```
Die Bären Café
```

HTMLENCODE

Converts a character in a string to its equivalent HTML character entity for the session encoding.



Hypertext Markup Language (HTML) contains various characters that are reserved in the language, such as quotation marks (" and ') and angle brackets (< and >). However, a Web page might need to display these characters. To enable this, the characters are replaced with `character entities`, which are codes that represent characters. For example, if a Web page contained the text:

```
value < 1 and > 7
```

the angle brackets would be interpreted as the delimiters for HTML tags, and the text would display incorrectly. To ensure the text displayed correctly the brackets can be replaced by named character entities (in this case, `<` and `>`).

Other characters can also be represented using entities; this is particularly useful for characters outside the 7-bit ASCII character set that you might want to represent in different session encodings. For example, the character ä can be represented by the entity `ä` .

Return type: Character

expression

Type: Character

A string containing characters to be converted.

options

Optional argument

Type: Character

There is one option, "7bit". This specifies that all characters (including reserved characters) in the 7-bit ASCII character remain unencoded, while characters outside of that set are encoded. By default, "7bit" is not set.

Note:

The option must be set as shown here; variations such as "7BIT" are not recognised and cause an error.

Characters are encoded by the function as hexadecimal number entities, except for the following characters, which are returned as named entities:

>	>
<	<
&	&
'	'
"	"

Basic example

In this example, the function converts character entities to equivalent characters. The result is written to the log.

```
DATA _NULL_;  
  length x $ 23;  
  x = HTMLENCODE("value < 1 and > 7");  
  PUT x;  
run;
```

This produces the following output:

```
value &lt; 1 and &gt; 7
```

Note:

The result of this example has been written to the log, but character entities would normally be written to a file containing HTML that is to be displayed on the Web.

Example – characters outside seven-bit set

In this example, the function converts characters outside the seven-bit ASCII character set to entities. The result is written to the log.

```
DATA _NULL_;

  length x $ 30;
  x = HTMLENCODE("Die Bären Café >", "7bit");
  PUT x;

  x = HTMLENCODE("Die Bären Café >");
  PUT x;

run;
```

This produces the following output:

```
Die B&#xE4;ren Caf&#xE9; >
Die Bären Café &gt;
```

In the first use of the function, the "7bit" option has been specified; the string returned contains entities in place of the characters that are not contained in the 7-bit ASCII character set; however, the character >, which is contained in the 7-bit ASCII character set, has not been encoded.

In the second use of the function, "7bit" is not specified, so characters in the 8-bit character set are returned unchanged, while the character > has been encoded.

Note:

The result of this example has been written to the log, but character entities would normally be written to a file containing HTML that is to be displayed on the Web.

URLDECODE

Returns the result of replacing escaped characters in a URL with the equivalent session character.

➤ **URLDECODE** ➤ (*string-to-decode*) ➤

A Uniform Resource Locator (URL) can only consist of characters from the ASCII character set. If a URL contains characters not in the ASCII character set, the characters are replaced by a code before being sent over the Internet. The code consists of an escape character (%) followed by two hexadecimal digits. There are also reserved characters, such as the space character, that also need to be converted. For example, in the URL `www.bicycle shop.com` the spaces would be replaced by the code %20, producing `www.bicycle%20shop.com`.

A URL obtained from the Internet or from another Web page might, therefore, contain one or more codes that need to be converted to equivalent characters.

Return type: Character

string-to-decode**Type:** Character

The URL to be decoded.

Example

In this example, the function converts the codes in a URL to their ASCII equivalent. The result is written to the log.

```
DATA _NULL_;  
    result = URLDECODE("www.steve%27s%20bicycles.com");  
    PUT result;  
run;
```

This produces the following output:

```
www.steve's bicycles.com
```

URLENCODE

Converts a non-ASCII or special character used in a URL to a code.

➤ **URLENCODE** ➤ (***string-to-encode***) ➤

A Uniform Resource Locator (URL) can only consist of characters from the ASCII character set. If a URL contains characters not in the ASCII character set, the characters are replaced by a code before being sent over the Internet. The code consists of an escape character (%) followed by two hexadecimal digits. There are also reserved characters, such as the space character, that also need to be converted. For example, in the URL `www.bicycle shop.com` the spaces would be replaced by the code %20, producing `www.bicycle%20shop.com`.

A URL you want to send across the Internet might, therefore, contain one or more characters that need replacing with an equivalent code.

Return type: Character***string-to-encode*****Type:** Character

The URL to encode.

Example

In this example, the function converts the special characters in the URL to corresponding codes. The result is written to the log.

```
DATA _NULL_;
  result = URLENCODE("www.steve's bicycles.com");
  PUT result;
run;
```

This produces the following output:

```
www.steve%27s%20bicycles.com
```

Note:

The result of this example has been written to the log, but the URL would normally be written to a file containing an HTML file for display on the Web, or sent across an HTTP connection.

Zipcode functions

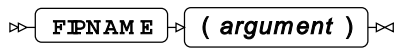
Accesses information in the ZIPCODE dataset and returns city names, state numbers, state codes, state names, and distances between locations.

FIPNAME ↗	2222
Returns the state name that corresponds to the specified FIPS state numeric code.	
FIPNAMEL ↗	2222
Returns the state name in sentence case that corresponds to the specified FIPS state numeric code.	
FIPSTATE ↗	2223
Returns the FIPS state alpha code that corresponds to the specified FIPS state numeric code.	
GEODIST ↗	2224
Returns the distance between two pairs of coordinates.	
STFIPS ↗	2226
Returns the FIPS state numeric code that corresponds to the specified FIPS state alpha code.	
STNAME ↗	2226
Returns the state name that corresponds to the specified FIPS state alpha code.	
STNAMEL ↗	2227
Returns the state name in sentence case that corresponds to the specified FIPS state alpha code.	
ZIPCITY ↗	2228
Returns the city name and FIPS state alpha code that corresponds to the specified ZIP code.	
ZIPCITYDISTANCE ↗	2228
Returns the distance between two cities in miles.	

ZIPFIPS ↗	2229
Returns the FIPS state numeric code that corresponds to the specified ZIP code.	
ZIPNAME ↗	2230
Returns the state name that corresponds to the specified ZIP code.	
ZIPNAMEL ↗	2230
Returns the state name in sentence case that corresponds to the specified ZIP code.	
ZIPSTATE ↗	2231
Returns the FIPS state alpha code that corresponds to the specified ZIP code.	

FIPNAME

Returns the state name that corresponds to the specified FIPS state numeric code.



Return type: Character

The name is returned in uppercase.

argument

Type: Numeric

The FIPS state numeric code.

Example

In this example, the state name that corresponds to the FIPS state numeric code 21 is returned. The result is written to the log.

```

DATA _NULL_;
  zipcd = FIPNAME(21);
  PUT "The state name is: " zipcd;
RUN;
  
```

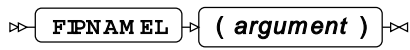
This produces the following output:

```

The state name is: KENTUCKY
  
```

FIPNAMEL

Returns the state name in sentence case that corresponds to the specified FIPS state numeric code.



Return type: Character

The name is returned in sentence case.

argument

Type: Numeric

The FIPS state numeric code.

Example

In this example, the state name that corresponds to the FIPS state numeric code 44 is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = FIPNAMEL(44);  
  PUT "The state name is: " zipcd;  
RUN;
```

This produces the following output:

```
The state name is: Rhode Island
```

FIPSTATE

Returns the FIPS state alpha code that corresponds to the specified FIPS state numeric code.

➤ **FIPSTATE** ➤ (*argument*) ➤

Return type: Character

argument

Type: Numeric

The FIPS state numeric code.

Example

In this example, the FIPS alpha state code that corresponds to the FIPS state numeric code 25 is returned. The result is written to the log.

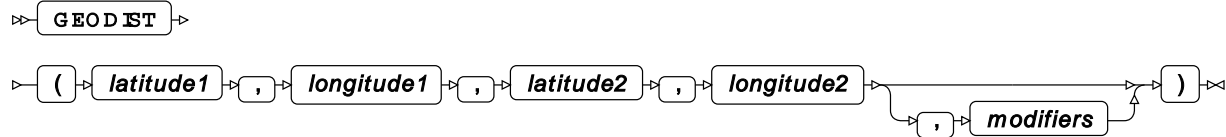
```
DATA _NULL_;  
  zipcd = FIPSTATE(25);  
  PUT "The FIPS state alpha code is: " zipcd;  
RUN;
```

This produces the following output:

```
The FIPS state alpha code is: MA
```

GEODIST

Returns the distance between two pairs of coordinates.



The function calculates the distance between two pairs of coordinates of latitude and longitude. If required, you can specify the units to be used (degrees or radians).

Return type: Numeric

The distance in kilometres or miles. By default kilometres are used.

latitude1

Type: Numeric

The latitude of the first location.

longitude1

Type: Numeric

The longitude of the first location.

latitude2

Type: Numeric

The latitude of the second location.

longitude2

Type: Numeric

The longitude of the second location.

modifiers

Optional argument

The unit used to specify latitude and longitude, and the unit in which the distance is returned, can be modified. You can, for example, specify that latitude and longitude are specified in radians, and the distance is returned in kilometers.

"D"

The values for latitude and longitude are specified in degrees. This is the default.

"K"

The value returned is the distance in kilometres. This is the default.

"M"

The value returned is the distance in miles.

"R"

The values for latitude and longitude are specified in radians.

Example – default distance unit

In this example, coordinates are specified for Ocean Shores in Washington State (46.970702,-124.150812) and Ocean Beach in New York State (40.647098,-73.151777). Because no values have been specified for *modifiers*, the distance is returned in the default unit (kilometres), and the specified latitude and longitude values are assumed to be in degrees. The result is written to the log.

```
DATA _NULL_;  
  dist = GEODIST(46.970702,-124.150812,40.647098,-73.151777);  
  PUT "The distance is: " dist "kilometres";  
RUN;
```

This produces the following output:

```
The distance is: 4089.0784652 kilometres
```

Example – specifying distance in miles

In this example, coordinates are specified for Ocean Shores in Washington State (46.970702,-124.150812) and Ocean Beach in New York State (40.647098,-73.151777). The **M** modifier is specified. The result is written to the log.

```
DATA _NULL_;  
  dist = GEODIST(46.970702,-124.150812,40.647098,-73.151777,"M");  
  PUT "The distance is: " dist "miles";  
RUN;
```

This produces the following output:

```
The distance is: 2540.8355601 miles
```

Example – specifying distance in radians

In this example, the **R** modifier is set, so the function assumes that latitudes and longitudes are specified in radians. The **M** modifier is also specified, so the distance is returned in miles. The coordinates in degrees for Ocean Shores in Washington State (46.970702,-124.150812) have been converted to radians (0.8197934, -2.1668404) before input, and similarly, the coordinates in degrees for Ocean Beach in New York State (40.647098,-73.151777) have been converted to radians (0.70942569, -1.27673936). The result is written to the log.

```
DATA _NULL_;  
  dist = GEODIST(0.8197934,-2.1668404,0.70942569,-1.27673936, "RM");  
  PUT "The distance is: " dist "miles";  
RUN;
```

This produces the following output:

```
The distance is: 2540.8354675 miles
```

STFIPS

Returns the FIPS state numeric code that corresponds to the specified FIPS state alpha code.

➤ **STFIPS** ➤ **(*argument*)** ➤

Return type: Numeric

argument

Type: Character

The FIPS state alpha code.

Example

In this example, the FIPS state numeric code corresponding to the specified FIPS state alpha code PR is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = STFIPS("PR");  
  PUT "The FIPS state numeric code is: " zipcd;  
RUN;
```

This produces the following output:

```
The FIPS state numeric code is: 72
```

STNAME

Returns the state name that corresponds to the specified FIPS state alpha code.

➤ **STNAME** ➤ **(*argument*)** ➤

Return type: Character

The name is returned in uppercase.

argument

Type: Character

The FIPS state alpha code.

Example

In this example, the state name corresponding to the specified FIPS state alpha code WV is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = STNAME("WV");  
  PUT "The state name is: " zipcd;  
RUN;
```

This produces the following output:

```
The state name is: WEST VIRGINIA
```

STNAMEL

Returns the state name in sentence case that corresponds to the specified FIPS state alpha code.

STNAMEL (*argument*)

The name is returned in sentence case.

Return type: Character

argument

Type: Character

The FIPS state alpha code.

Example

In this example, the state name corresponding to the specified FIPS state alpha code NC is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = STNAMEL("NC");  
  PUT "The state name is: " zipcd;  
RUN;
```

This produces the following output:

```
The state name is: North Carolina
```

ZIPCITY

Returns the city name and FIPS state alpha code that corresponds to the specified ZIP code.

➤ `ZIPCITY` ➤ `(zipcode)` ➤

Return type: Character

zipcode

Type: Character or numeric value

The ZIP code.

Example

In this example, the city name and FIPS state alpha code corresponding to the specified ZIP code is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = ZIPCITY(10017);  
  PUT "The city and FIPS state alpha code are: " zipcd;  
RUN;
```

This produces the following output:

```
The city and FIPS state alpha code are: New York, NY
```

ZIPCITYDISTANCE

Returns the distance between two cities in miles.

➤ `ZIPCITYDISTANCE` ➤ `(zipcode1 , zipcode2)` ➤

Return type: Numeric

zipcode1

Type: Character or numeric value

The ZIP code for the first city.

zipcode2

Type: Character or numeric value

The ZIP code for the second city.

Example

In this example, the ZIP codes for New York (10017) and Anasco (00610) are used to calculate the distance in miles between these cities. The result is written to the log.

```
DATA _NULL_;  
  dist = ZIPCITYDISTANCE(10017, 00610);  
  PUT "The distance is: " dist "miles";  
RUN;
```

This produces the following output:

```
The distance is: 1599.7901691 miles
```

ZIPFIPS

Returns the FIPS state numeric code that corresponds to the specified ZIP code.

➤ **ZIPFIPS** ➤ (*zipcode*) ➤

Return type: Numeric

zipcode

Type: Character or numeric value

The ZIP code.

Example

In this example, the FIPS state numeric code corresponding to the specified ZIP code is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = ZIPFIPS(55041);  
  PUT "The FIPS state numeric code is: "  
RUN;
```

This produces the following output:

```
The FIPS state numeric code is: 27
```

In this case, the ZIP code covers an area in South Minnesota, for which the FIPS state numeric code is 27.

ZIPNAME

Returns the state name that corresponds to the specified ZIP code.

➤ `ZIPNAME` ➤ `(zipcode)` ➤

Return type: Character

The name is returned in uppercase.

zipcode

Type: Character or numeric value

The ZIP code.

Example

In this example, the state name corresponding to the specified ZIP code is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = ZIPNAME(60140);  
  PUT "The state name is: " zipcd;  
RUN;
```

This produces the following output:

```
The state name is: ILLINOIS
```

In this case, the ZIP code covers an area in North East Illinois.

ZIPNAMEL

Returns the state name in sentence case that corresponds to the specified ZIP code.

➤ `ZIPNAMEL` ➤ `(zipcode)` ➤

Return type: Character

The name is returned in sentence case.

zipcode

Type: Character or numeric value

The ZIP code.

Example

In this example, the state name corresponding to the specified ZIP code is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = ZIPNAMEL(20016);  
  PUT "The state name is: " zipcd;  
RUN;
```

This produces the following output:

```
The state name is: District of Columbia
```

ZIPSTATE

Returns the FIPS state alpha code that corresponds to the specified ZIP code.

➤ **ZIPSTATE** ➤ (*zipcode*) ➤

Return type: Character

zipcode

Type: Character or numeric value

The ZIP code.

Example

In this example, the FIPS state alpha code corresponding to the specified ZIP code is returned. The result is written to the log.

```
DATA _NULL_;  
  zipcd = ZIPSTATE(10017);  
  PUT "The FIPS state alpha code is: " zipcd;  
RUN;
```

This produces the following output:

```
The FIPS state alpha code is: NY
```

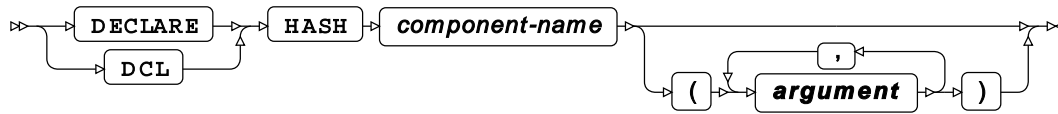
DATA step Components

HASH Component

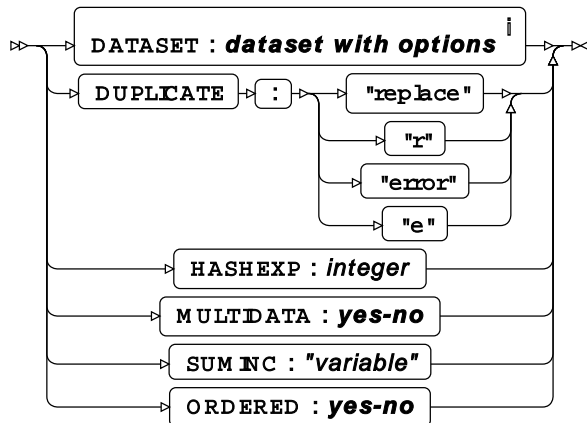
Supported statements

- *HASH* [↗](#) (page 2233)
- *ADD* [↗](#) (page 2233)
- *CHECK* [↗](#) (page 2233)
- *CLEAR* [↗](#) (page 2234)
- *DECLARE* [↗](#) (page 2234)
- *DEFINEDATA* [↗](#) (page 2234)
- *DEFINEDONE* [↗](#) (page 2234)
- *DEFINEKEY* [↗](#) (page 2234)
- *DELETE* [↗](#) (page 2234)
- *EQUALS* [↗](#) (page 2235)
- *FIND* [↗](#) (page 2235)
- *FIND_NEXT* [↗](#) (page 2235)
- *FIND_PREV* [↗](#) (page 2235)
- *HAS_NEXT* [↗](#) (page 2235)
- *HAS_PREV* [↗](#) (page 2235)
- *ITEM_SIZE* [↗](#) (page 2235)
- *NUM_ITEMS* [↗](#) (page 2236)
- *OUTPUT* [↗](#) (page 2236)
- *REF* [↗](#) (page 2236)
- *REMOVE* [↗](#) (page 2236)
- *REMOVEDUP* [↗](#) (page 2236)
- *REPLACE* [↗](#) (page 2236)
- *REPLACEDUP* [↗](#) (page 2237)
- *SUM* [↗](#) (page 2237)
- *SUMDUP* [↗](#) (page 2237)

HASH

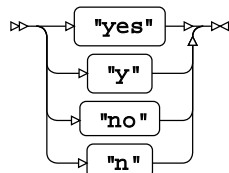


argument

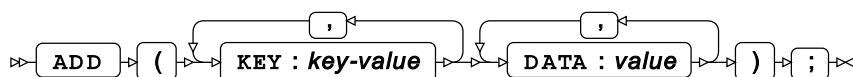


ⁱ See [Input dataset](#) (page 16).

yes-no



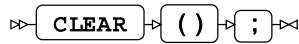
ADD



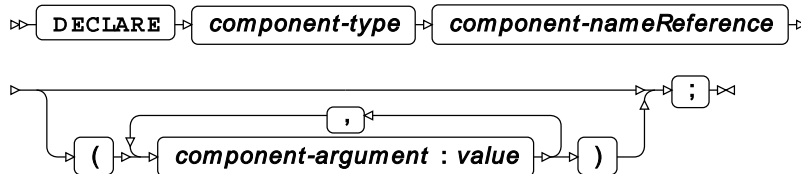
CHECK



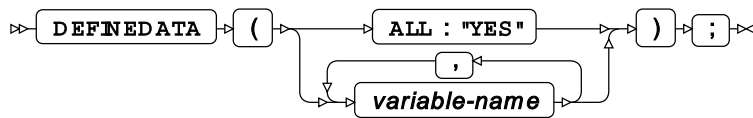
CLEAR



DECLARE



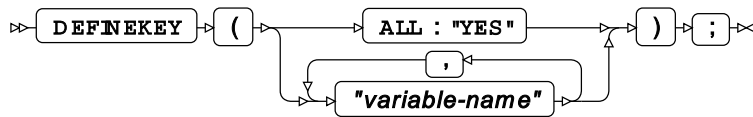
DEFNEDATA



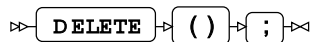
DEFNEDONE



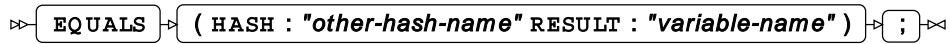
DEFINEKEY



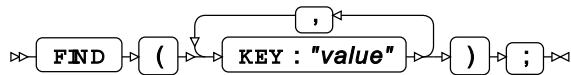
DELETE



EQUALS



FIND



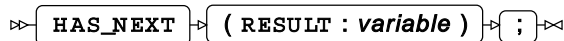
FIND_NEXT



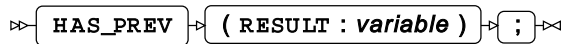
FIND_PREV



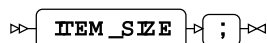
HAS_NEXT



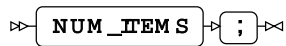
HAS_PREV



ITEM_SIZE



NUM_ITEMS

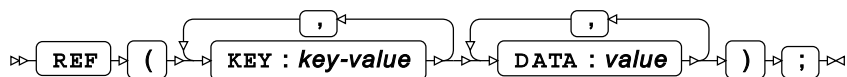


OUTPUT

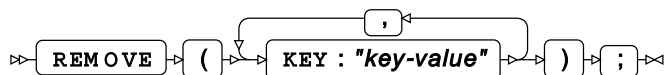


ⁱ See [Dataset](#) (page 16).

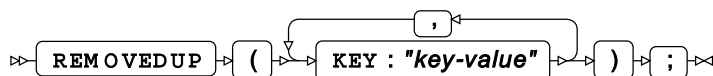
REF



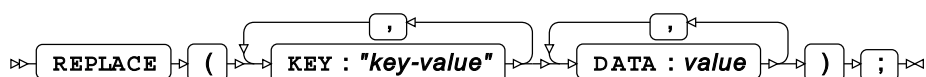
REMOVE



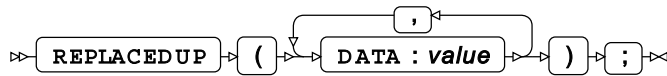
REMOVEDUP



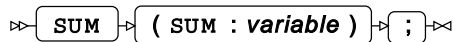
REPLACE



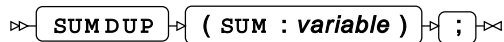
REPLACEDUP



SUM



SUMDUP

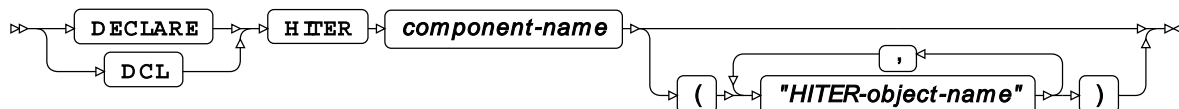


HITER Component

Supported statements

- *HITER* [↗](#) (page 2237)
- *FIRST* [↗](#) (page 2238)
- *LAST* [↗](#) (page 2238)
- *NEXT* [↗](#) (page 2238)
- *PREV* [↗](#) (page 2238)
- *SUM* [↗](#) (page 2238)

HITER



FIRST

➤ **FIRST** ➤ () ➤ ; ➤

LAST

➤ **LAST** ➤ () ➤ ; ➤

NEXT

➤ **NEXT** ➤ () ➤ ; ➤

PREV

➤ **PREV** ➤ () ➤ ; ➤

SUM

➤ **SUM** ➤ (SUM : *variable*) ➤ ; ➤

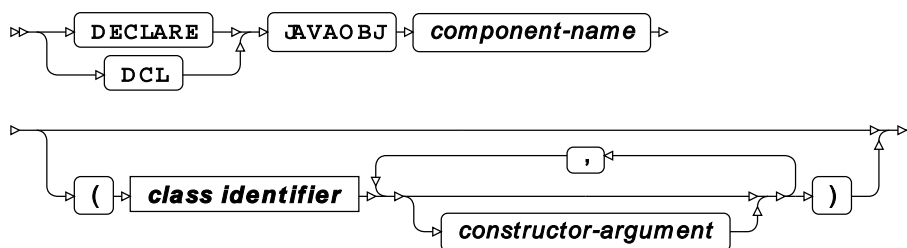
JAVAOBJ Component

Supported statements

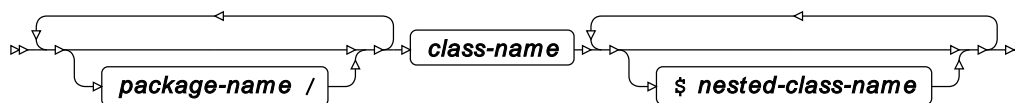
- [JAVAOBJ](#) (page 2239)
- [CALLSTATICtypeMETHOD](#) (page 2240)
- [CALLtypeMETHOD](#) (page 2240)
- [CALLSTATICVOIDMETHOD](#) (page 2240)
- [CALLVOIDMETHOD](#) (page 2240)
- [EXCEPTIONCHECK](#) (page 2241)
- [EXCEPTIONCLEAR](#) (page 2241)

- [EXCEPTIONDESCRIBE](#) (page 2241)
- [FLUSHJAVAOUTPUT](#) (page 2241)
- [GETSTATICtypeFIELD](#) (page 2241)
- [GETtypeFIELD](#) (page 2242)
- [SETSTATICtypeFIELD](#) (page 2242)
- [SETtypeFIELD](#) (page 2242)

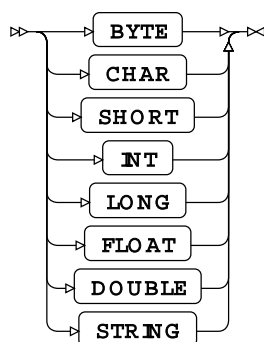
JAVAOBJ



class identifier

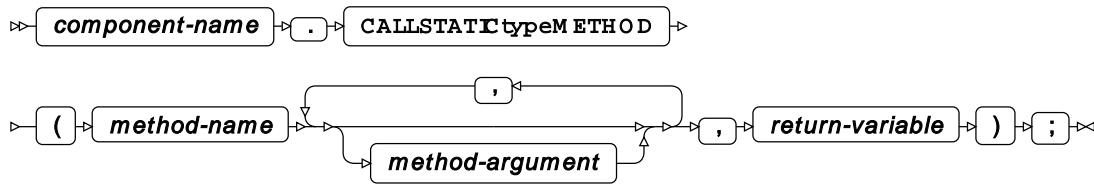


JAVAOBJ type



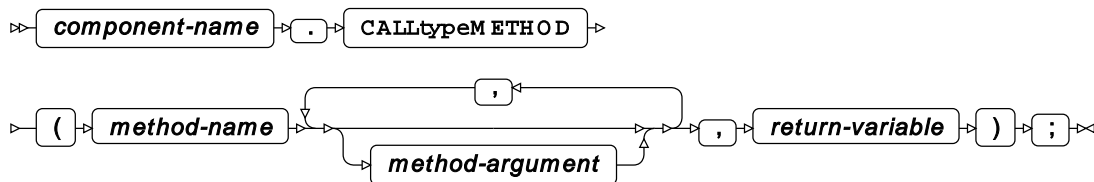
Substitute term *type* by one of these types in the component names that follow.

CALLSTATICtypeMETHOD



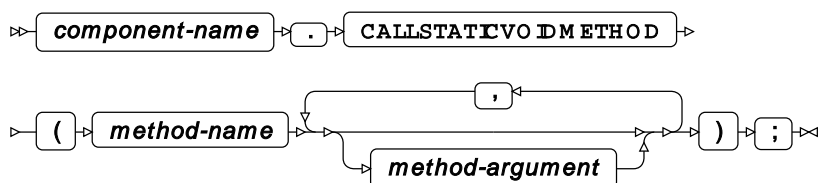
The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

CALLtypeMETHOD



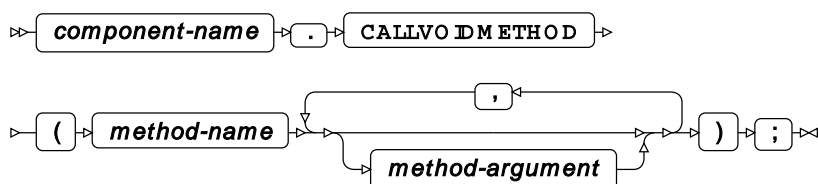
The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

CALLSTATICVOIDMETHOD



The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

CALLVOIDMETHOD



The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

EXCEPTIONCHECK

```
component-name . EXCEPTIONCHECK ( numeric-variable ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

EXCEPTIONCLEAR

```
component-name . EXCEPTIONCLEAR ( ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

EXCEPTIONDESCRIBE

```
component-name . EXCEPTIONDESCRIBE ( boolean-value ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

FLUSHJAVAOUTPUT

```
component-name . FLUSHJAVAOUTPUT ( ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

GETSTATICtypeFIELD

```
component-name . GETSTATICtypeFIELD ( field-name , return-variable ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

GETtypeFIELD

```
component-name . GETtypeFIELD ( field-name , return-variable ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

SETSTATICtypeFIELD

```
component-name . SETSTATICtypeFIELD ( field-name , value ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

SETtypeFIELD

```
component-name . SETtypeFIELD ( field-name , value ) ;
```

The *component-name* variable used with this statement is created using the `DECLARE JAVAOBJ` statement. See [JAVAOBJ](#) (page 2239)

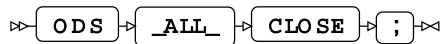
Output Delivery System

The Output Delivery System(ODS) routes the contents of program output to the sources described in this section, including HTML and PDF.

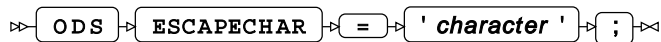
ODS global statements

Statements that affect all output destinations.

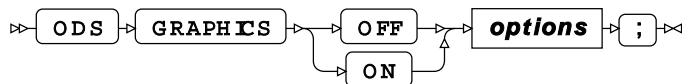
ODS _ALL_ CLOSE



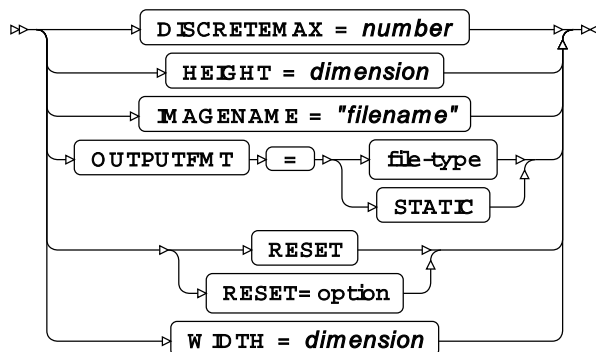
ODS ESCAPECHAR



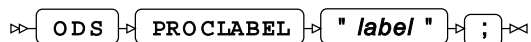
ODS GRAPHICS



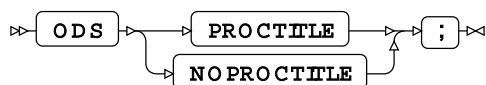
options



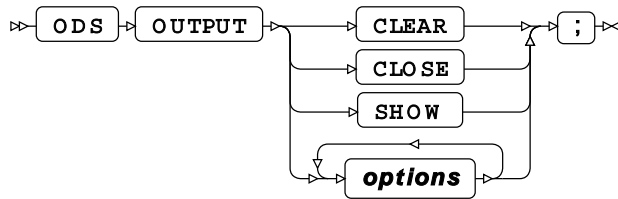
ODS PROCLABEL



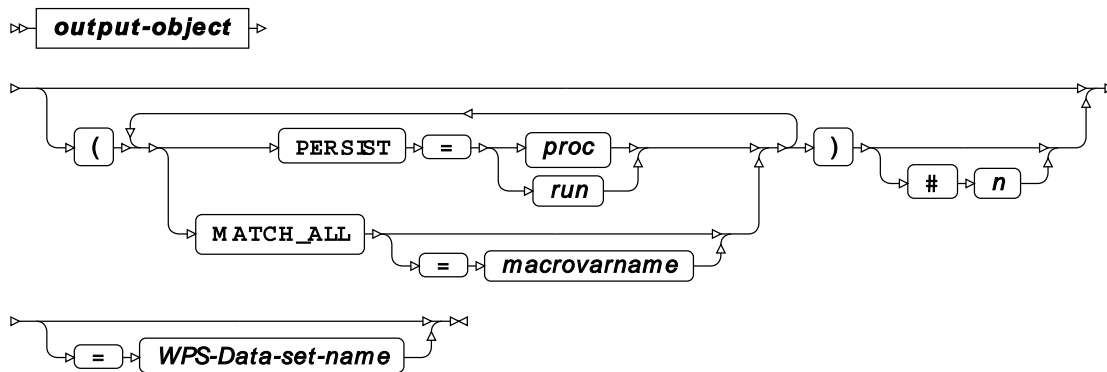
ODS PROCTITLE



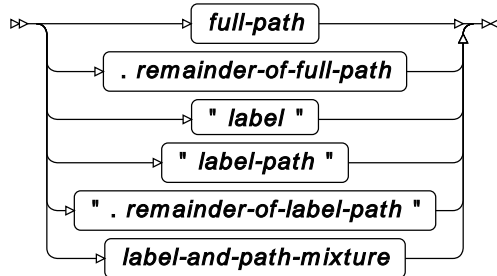
ODS OUTPUT



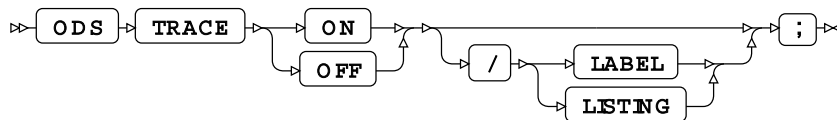
options



output-object

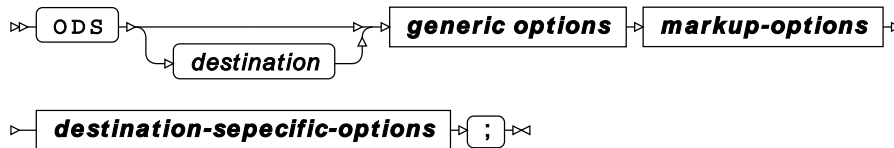


ODS TRACE

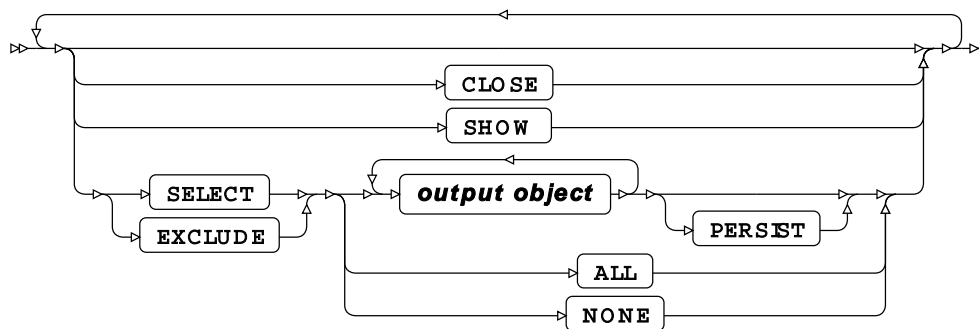


ODS

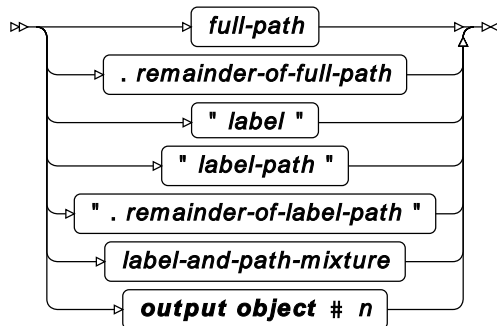
The generic structure of an ODS global statement is shown below. Subsequent sections detail the use of the ODS system for specific destinations.



generic options

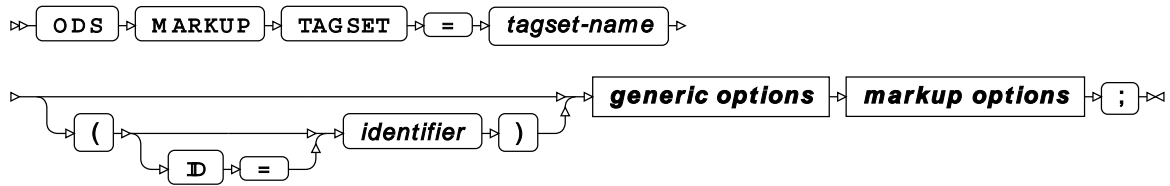


output object

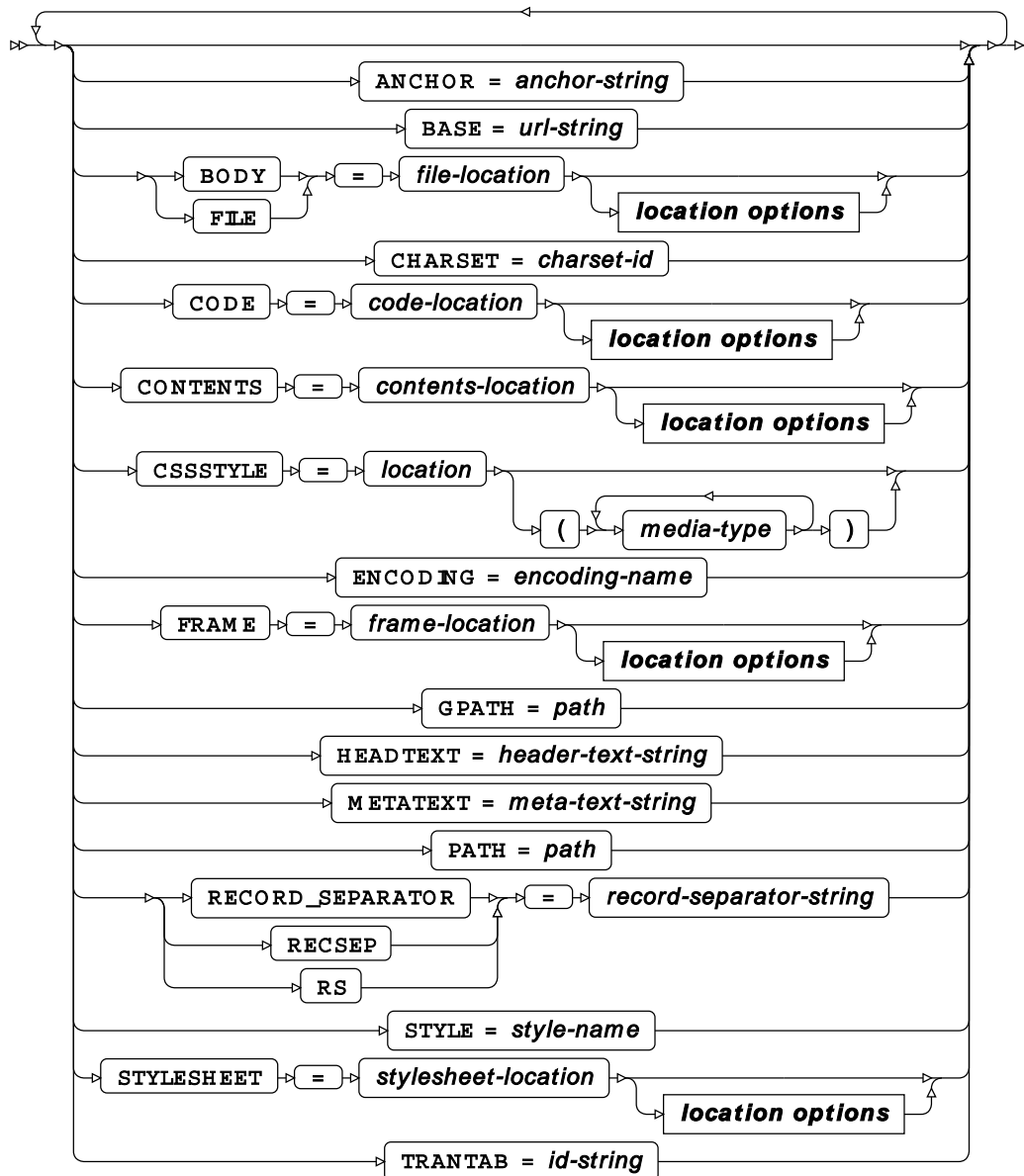


ODS MARKUP

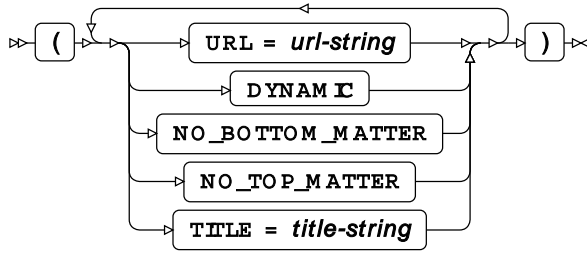
Below is a generic list of markup options supported by ODS. Each destination accepts a subset of these options as specified in the corresponding sections.



markup options



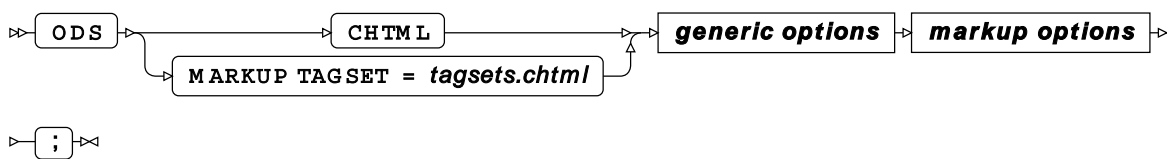
location options



The following statements are aliases of `ODS MARKUP`. They are described in the corresponding sections in further detail.

<code>ODS CHTML</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.CHTML</code>
<code>ODS CSV</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.CSV</code>
<code>ODS CSVALL</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.CSVALL</code>
<code>ODS EXCELXP</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.EXCELXP</code>
<code>ODS HTML</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.HTMLCSS</code>
<code>ODS HTMLCSS</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.HTMLCSS</code>
<code>ODS HTML4</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.HTML4</code>
<code>ODS MSOFFICE2K</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.MSOFFICE2K</code>
<code>ODS PHTML</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.PHTML</code>
<code>ODS XML</code>	is an alias of <code>ODS MARKUP TAGSET=TAGSETS.XML</code>

ODS CHTML

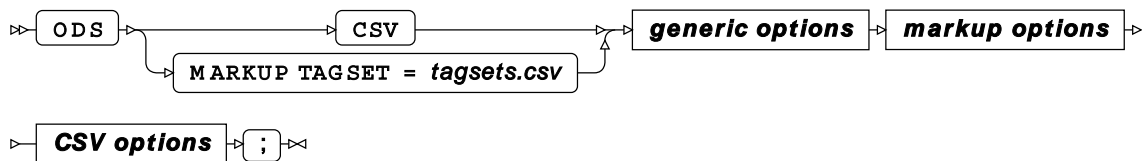


The following markup options are supported. The syntax for these options is shown in *markup options* in the section [ODS MARKUP](#) (page 2245):

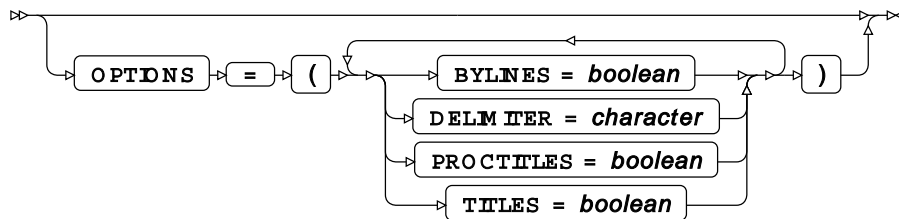
- ANCHOR
- BODY
- CHARSET
- CONTENTS
- ENCODING
- FILE
- FRAME

- GPATH
- PAGE
- PATH
- RECORD_SEPARATOR
- TRANTAB

ODS CSV



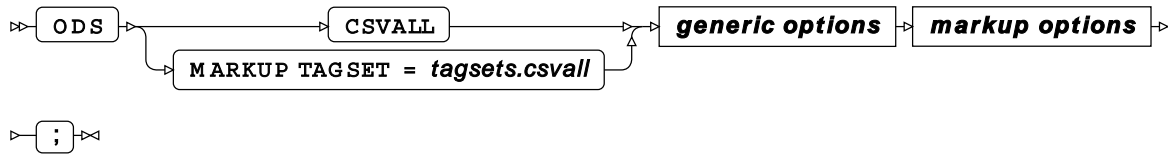
CSV options



The following markup options are supported. The syntax for these options is shown in *markup options* in the section [ODS MARKUP](#) (page 2245):

- BODY
- CHARSET
- ENCODING
- FILE
- GPATH
- PATH
- RECORD_SEPARATOR
- TRANTAB

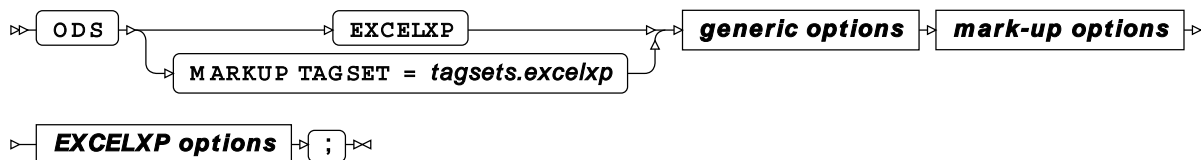
ODS CSVALL



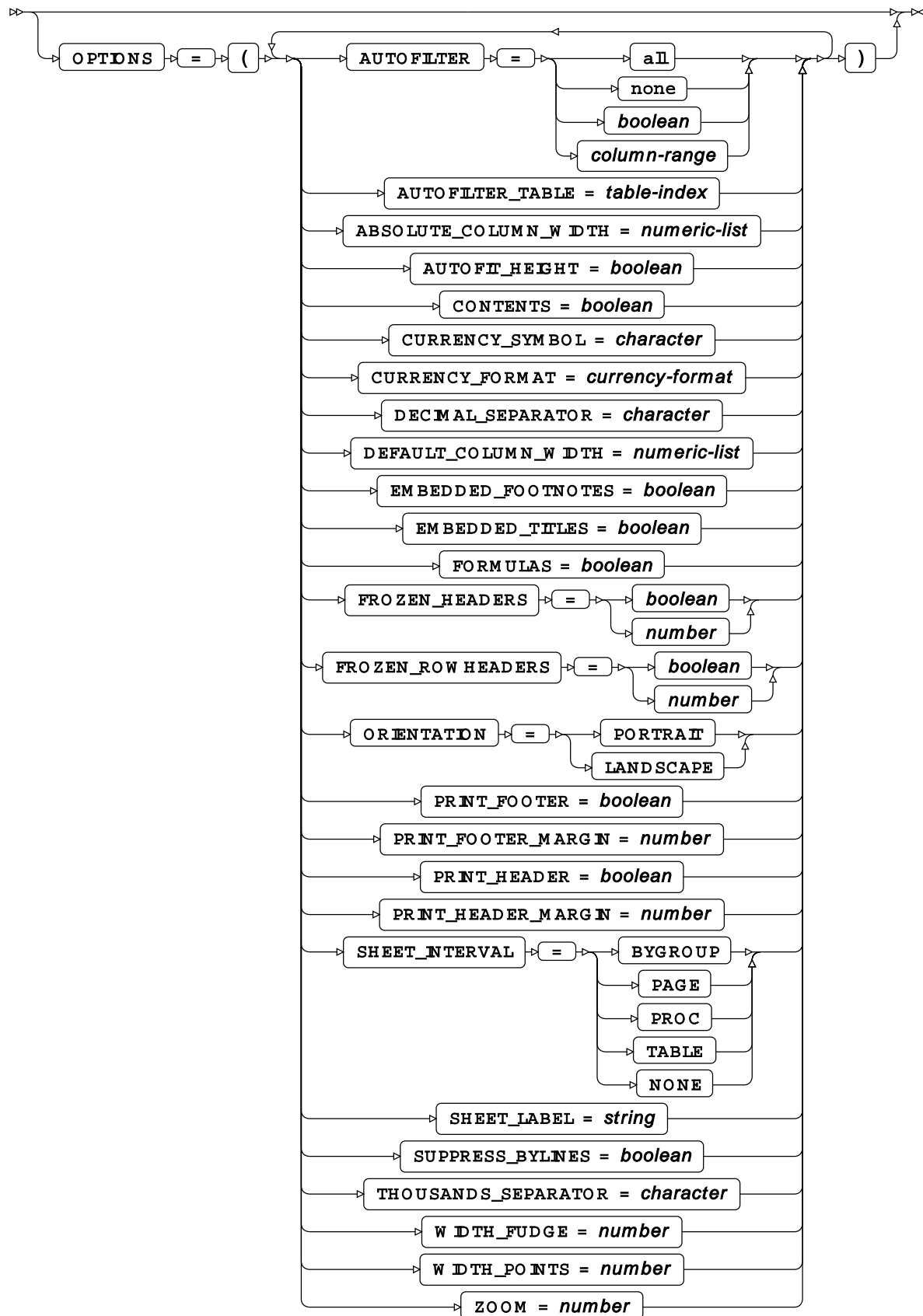
The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- BODY
- CHARSET
- ENCODING
- FILE
- GPATH
- PATH
- RECORD_SEPARATOR
- TRANTAB

ODS EXCELPX



EXCELXP options



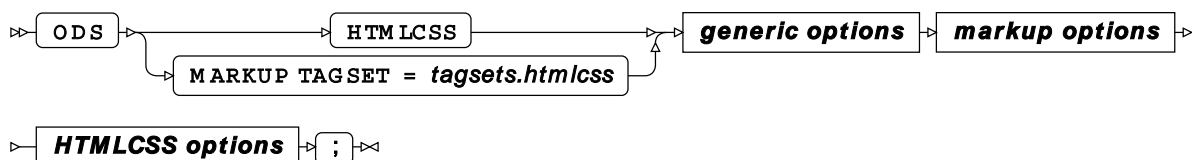
The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- BODY
- CHARSET
- CONTENTS
- ENCODING
- FILE
- GPATH
- PAGE
- PATH
- RECORD_SEPARATOR
- STYLE
- STYLESHEET
- TRANTAB

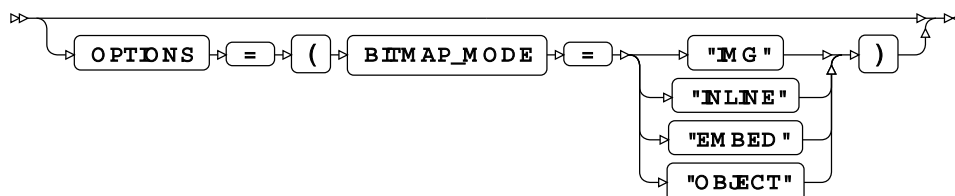
ODS HTML

The destination `HTML` is an alias for `HTMLCSS`, see section *ODS HTMLCSS* [↗](#) (page 2251).

ODS HTMLCSS



HTMLCSS options

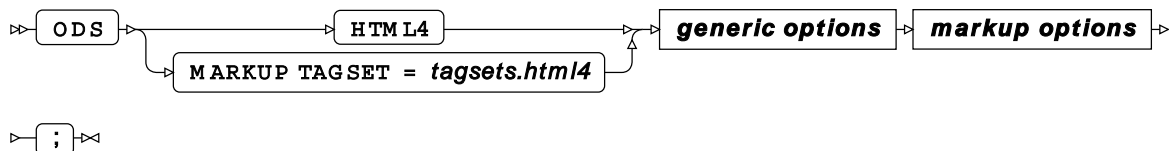


The following markup options are supported (see section *ODS MARKUP* [↗](#) (page 2245)):

- ANCHOR
- BODY

- CHARSET
- CONTENTS
- CSSSTYLE
- ENCODING
- FILE
- FRAME
- GPATH
- HEADTEXT
- METATEXT
- PAGE
- PATH
- RECORD_SEPARATOR
- STYLE
- STYLESHEET
- TRANTAB

ODS HTML4



The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- BODY
- CHARSET
- ENCODING
- FILE
- GPATH
- PATH
- RECORD_SEPARATOR
- TRANTAB

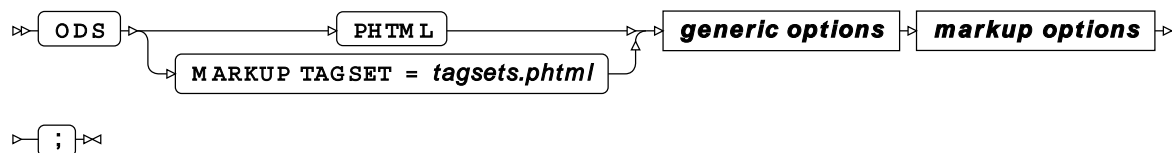
ODS MSOFFICE2K



The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- ANCHOR
- BODY
- CHARSET
- CONTENTS
- CSSSTYLE
- ENCODING
- FILE
- FRAME
- GPATH
- HEADTEXT
- METATEXT
- PAGE
- PATH
- RECORD_SEPARATOR
- STYLE
- STYLESHEET
- TRANTAB

ODS PHTML

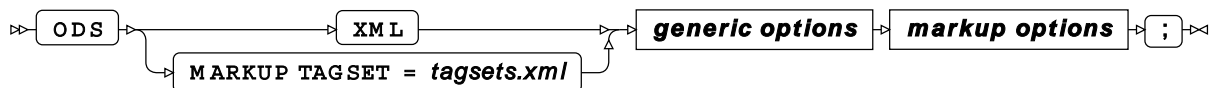


The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- BODY

- CHARSET
- ENCODING
- FILE
- GPATH
- HEADTEXT
- METATEXT
- PATH
- RECORD_SEPARATOR
- STYLE
- TRANTAB

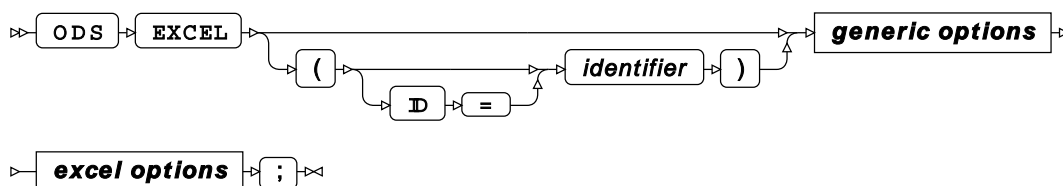
ODS XML



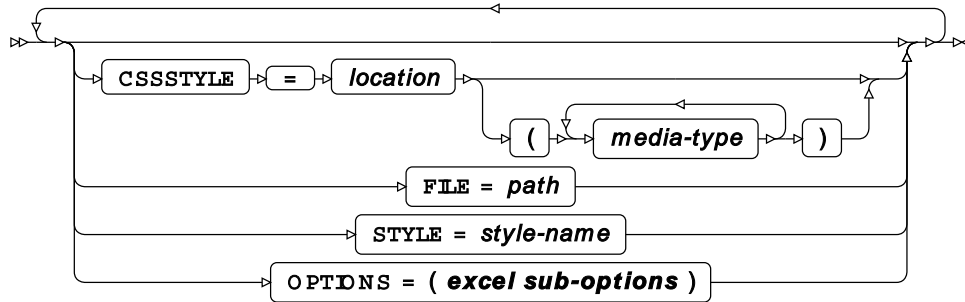
The following markup options are supported. The syntax for these options is shown in *markup options* in the section *ODS MARKUP* [↗](#) (page 2245):

- BODY
- CHARSET
- CONTENTS
- ENCODING
- FILE
- GPATH
- PATH
- RECORD_SEPARATOR
- TRANTAB

ODS EXCEL



excel options



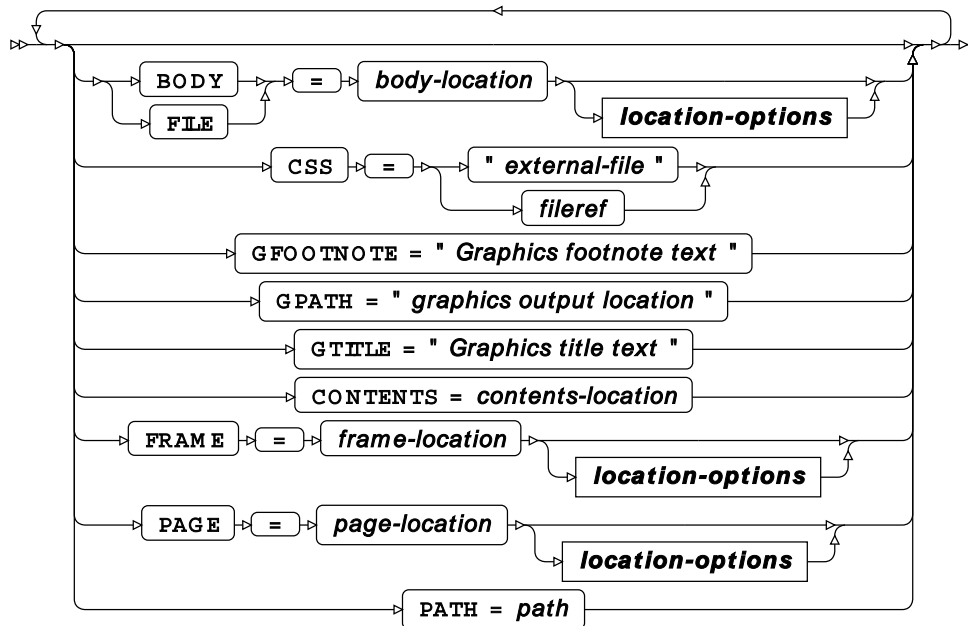
excel sub-options



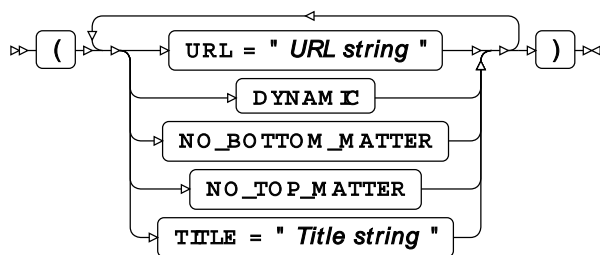
ODS OLDHTML



OLDHTML options



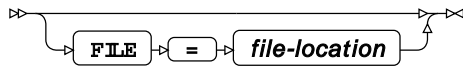
location-options



ODS LISTING



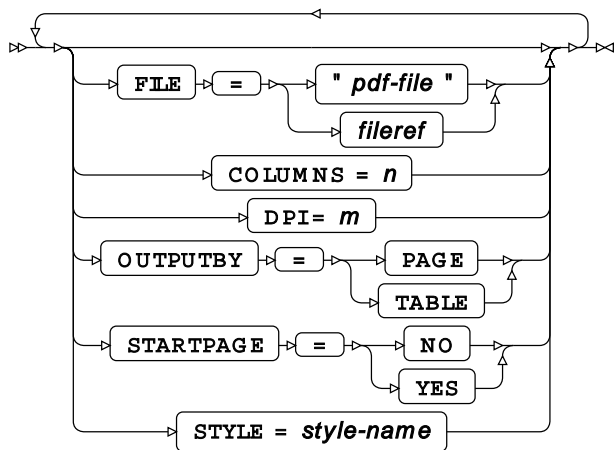
LISTING options



ODS PDF



PDF options



ODS PDF options

The following system options allow you to determine how PDFs are output via the ODS:

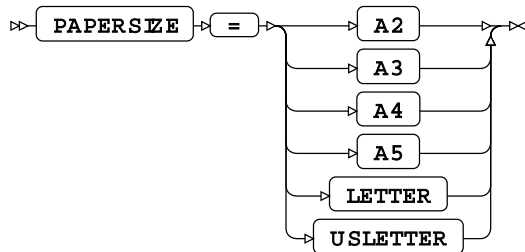
- *PAPER SIZE* [↗](#) (page 2259)
- *ORIENTATION* [↗](#) (page 2259)
- *TOPMARGIN* [↗](#) (page 2259)
- *BOTTOMMARGIN* [↗](#) (page 2260)
- *LEFTMARGIN* [↗](#) (page 2260)
- *RIGHTMARGIN* [↗](#) (page 2260)

Note:

Negative values are not allowed.

PAPER SIZE

This option sets the paper size of the PDFs output via the ODS.

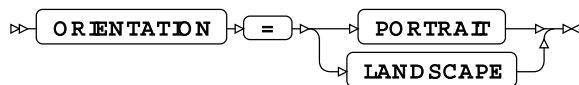


Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: The paper size associated with the Locale.

ORIENTATION

This option sets the orientation (PORTRAIT or LANDSCAPE) of the PDFs output via the ODS.

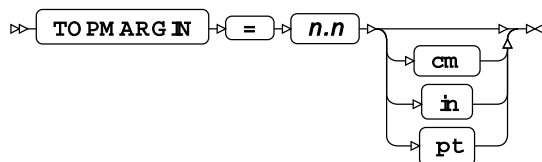


Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: PORTRAIT

TOPMARGIN

This option sets the top margin of the of the PDFs output via the ODS.

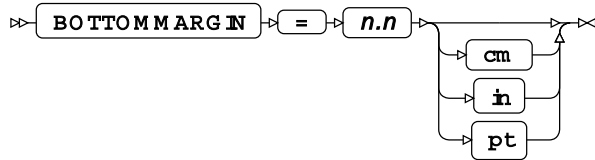


Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: Zero inches

BOTTOMMARGIN

This option sets the bottom margin of the of the PDFs output via the ODS.

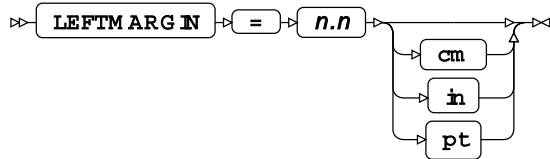


Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: Zero inches

LEFTMARGIN

This option sets the left margin of the of the PDFs output via the ODS.

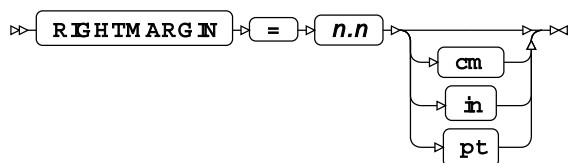


Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: Zero inches

RIGHTMARGIN

This option sets the right margin of the of the PDFs output via the ODS.



Valid in: ODS PDF: in configuration file, Options statement and/or command line

Default: Zero inches

Procedures

A fundamental set of procedures which provides a range of access, data manipulation, and basic statistical analysis.

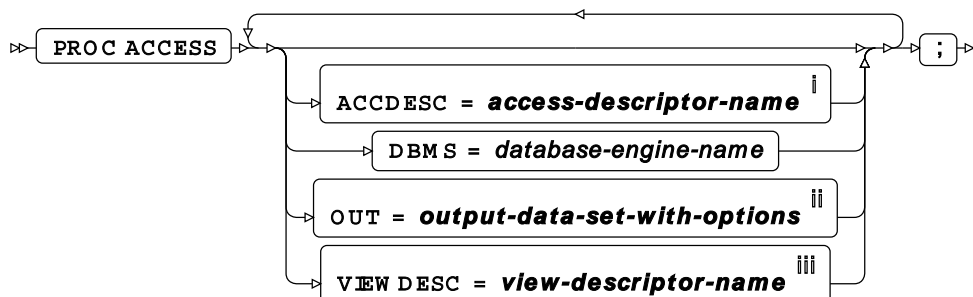
ACCESS procedure

Supported statements

- *PROC ACCESS* [↗](#) (page 2261)
- *ASSIGN* [↗](#) (page 2262)
- *CREATE* [↗](#) (page 2262)
- *DROP* [↗](#) (page 2262)
- *FORMAT* [↗](#) (page 2262)
- *LIST* [↗](#) (page 2263)
- *RENAME* [↗](#) (page 2263)
- *RESET* [↗](#) (page 2263)
- *SELECT* [↗](#) (page 2263)
- *SUBSET* [↗](#) (page 2264)
- *TABLE* [↗](#) (page 2264)
- *UNIQUE* [↗](#) (page 2264)

PROC ACCESS

Manages access descriptors and view descriptors.



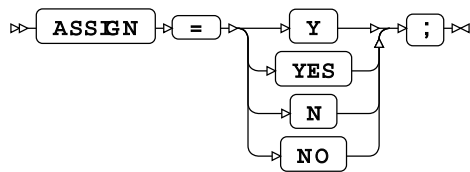
ⁱ See *Access Descriptors* [↗](#) (page 15).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

iii See *View Descriptors* [↗](#) (page 15).

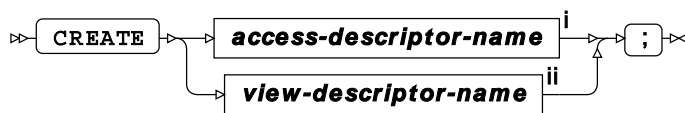
ASSIGN

Specifies whether to automatically assign names to variables in an access descriptor.



CREATE

Creates a new access descriptor or view descriptor.

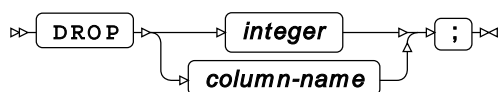


ⁱ See *Access Descriptors* [↗](#) (page 15).

ⁱⁱ See *View Descriptors* [↗](#) (page 15).

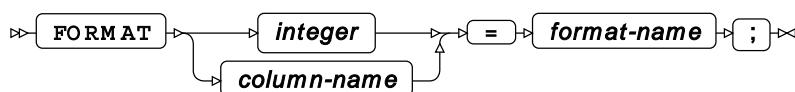
DROP

Drops one or more variables from an access descriptor.



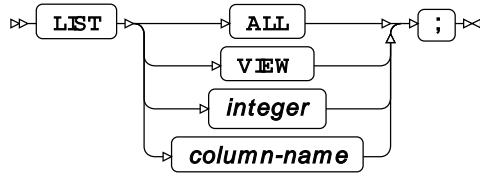
FORMAT

Applies formats to variables in an access descriptor.



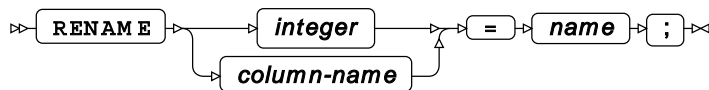
LIST

Prints out the description of one or more access descriptors or view descriptors.



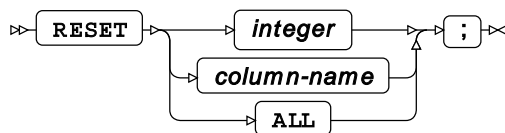
RENAME

Renames one or more variables in an access descriptor.



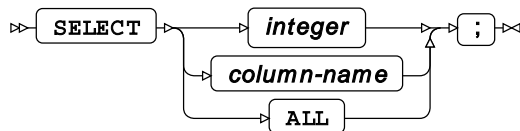
RESET

Resets to the default state any modification made to variables in an access or view descriptor.



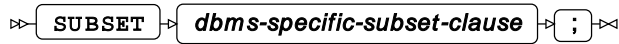
SELECT

Selects the variables to use in a view descriptor.



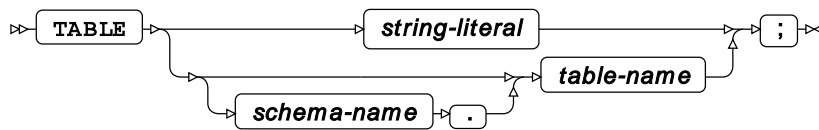
SUBSET

Applies a subset clause to a view descriptor.



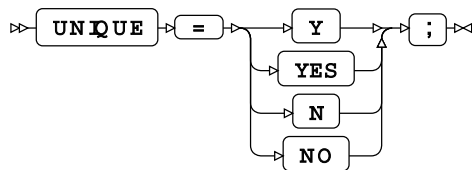
TABLE

Specifies a database table to which an access descriptor has been applied.



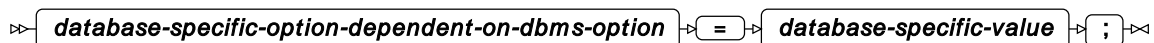
UNIQUE

Specifies whether the names of the variables in the access descriptor must be unique.



Connection option

Lists connection options for the specified DBMS. These options include username, password, and so on.



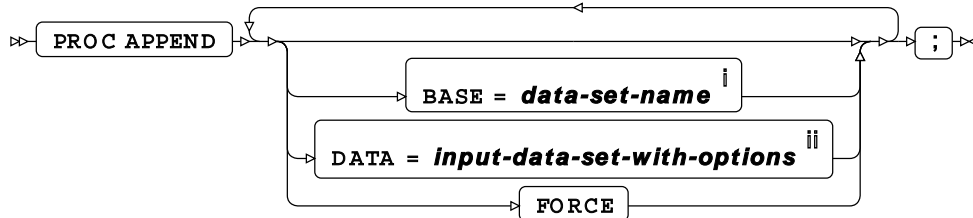
APPEND procedure

Supported statements

- `PROC APPEND` [↗](#) (page 2265)
- `WHERE` [↗](#) (page 2265)

PROC APPEND

Appends the observations from one dataset to another dataset.

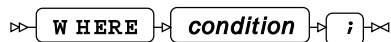


ⁱ See *Dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

WHERE

Restricts the observations to be processed.



APPSRV Procedure

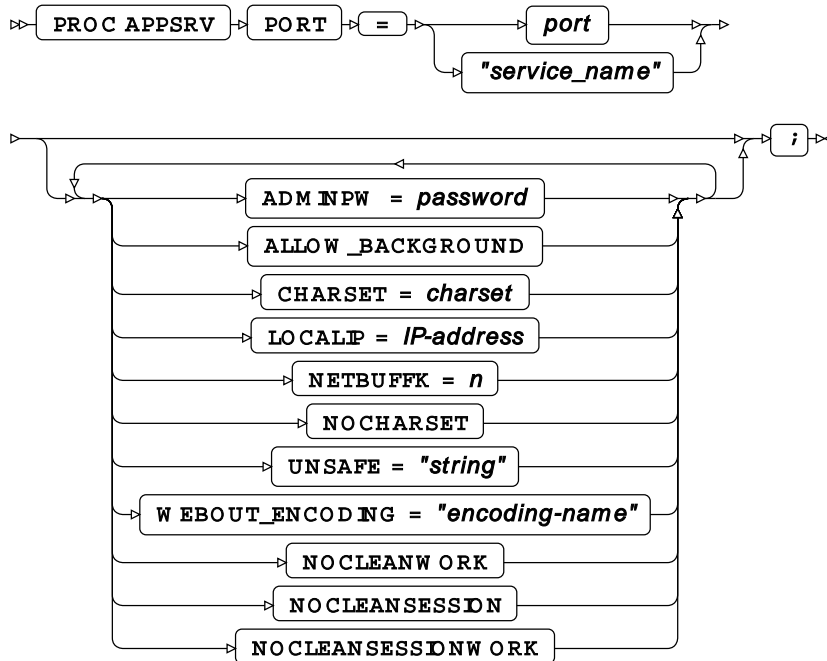
The `APPSRV` procedure configures and launches an Application Server.

Supported statements

- `PROC APPSRV` [↗](#) (page 2266)
- `ADMINLIBS` [↗](#) (page 2270)
- `ALLOCATE` [↗](#) (page 2271)
- `DATALIBS` [↗](#) (page 2271)
- `LOG` [↗](#) (page 2272)
- `PROGLIBS` [↗](#) (page 2274)
- `SESSIONLIBS` [↗](#) (page 2280)
- `SESSIONWORKLIBS` [↗](#) (page 2280)
- `REQUEST` [↗](#) (page 2275)
- `STATISTICS` [↗](#) (page 2280)
- `REQUEST` [↗](#) (page 2275)

- [WORKLIBS](#) (page 2283)

PROC APPSRV



ADMINPW

The `ADMINPW` option specifies the password that will be required to run programs in the `ADMINLIBS` libraries and special administrative programs such as `stop`. To avoid displaying a plain text password in the source code of an Application Server launch program, the `wpswebpassword` program should be used to generate an obfuscated version to be used as the `ADMINPW` option.

The following example shows how to set the `ADMINPW` option, albeit with a plain text password:

```
PROC APPSRV PORT=5001 ADMINPW="password";
  ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';
  PROGLIBS demo;
RUN;
```

ALLOW_BACKGROUND

The `ALLOW_BACKGROUND` option enables Application Servers that are members of a pool service to be placed in a background state. If this option is not present, attempts to place an Application Server into a background state will fail.

This example shows how to set the `ALLOW_BACKGROUND` option:

```
PROC APPSRV PORT=5001 ALLOW_BACKGROUND;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

CHARSET

The `CHARSET` option is used to pass the encoding type applied to the HTTP content-type header as generated by the WPS Web automatic header system. If the `CHARSET` option is not set, it will be set to the same encoding as is applied to the `_WEBOUT` fileref. If, rather than the `CHARSET` option, the `NOCHARSET` option is set, then no encoding information will be written by the automatic header system. `CHARSET` and `NOCHARSET` cannot both be set at the same time.

This example sets the `CHARSET` option to `latin1`:

```
PROC APPSRV PORT=5001 CHARSET='latin1';  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

With the above `APPSRV` invocation, the automatic header system might emit an HTTP header formed as below:

```
Content-type: text/html; charset='latin1'
```

LOCALIP

The `LOCALIP` option can be used to specify the IP address to which the Application Server should bind when it starts up. This is useful when a host has multiple addresses assigned to it - a situation known as a multi-homed host.

This example launches an Application Server that binds to the socket address `192.168.0.1:5001`:

```
PROC APPSRV PORT=5001 LOCALIP=192.168.0.1;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

NETBUFFK

The `NETBUFFK` option specifies the number of kilobytes an Application Server should reserve for a net buffer. A net buffer is used to buffer the number of data transfers between the server and the Broker as a program generates output. If the option is not specified, the Application Server will not use a net buffer.

Validation

$4 \leq \text{NETBUFFK} \leq 128$

The following APPSRV invocation specifies an 8K net buffer:

```
PROC APPSRV PORT=5001 NETBUFFK=8;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

NOCHARSET

The `NOCHARSET` option is used to suppress the generation of an encoding type by the automatic header system.

This example sets the `NOCHARSET` option:

```
PROC APPSRV PORT=5001 NOCHARSET;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

With the above APPSRV invocation, the automatic header system might emit an HTTP header formed as below:

```
Content-type: text/html
```

NOCLEANSESSION

The `NOCLEANSESSION` option is used to prevent the cleanup of the `SAVE` libraries which are created for any session generated by the Application Server. This option is useful for debugging but should not be used in production systems.

NOCLEANSESSIONWORK

The `NOCLEANSESSIONWORK` option is used to prevent the cleanup of the `WORK` library used when a `SESSION INIT` or `TERM` program is executed (refer to the `INIT` and `TERM` options on the `SESSION` statement).

NOCLEANWORK

The `NOCLEANWORK` option is used to prevent the cleanup of the `WORK` library which is allocated when a program is executed in response to a WPS Web request.

PORT (required)

The `PORT` option specifies the TCP port number that the Application Server should bind to when opening a communication socket. This can be provided as a number or a service name as specified in the `/etc/services` file on most UNIX-style operating systems.

Validation

`1 <= PORT <= 65535`

Note that some operating systems require special user privileges to use port numbers below 1024. The following example starts an Application Server bound to port 5001:

```
PROC APPSRV PORT=5001;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

By contrast, the next example starts an Application Server bound to the port number associated with the `wpsweb` service definition:

```
PROC APPSRV PORT='wpsweb';  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

If the port number is not specified or the service name cannot be resolved to a port number, the Application Server will fail to start and an error will be returned.

UNSAFE

The `UNSAFE` option is used to filter unsafe characters from user input. Parameters are passed as macro variables to the program to be executed, and to reduce the opportunities for a code-injection attack, unsafe characters are removed from parameter values before they are assigned to macro-variable values (although the raw value can always be accessed via the `APPSRV_UNSAFE()` function).

The default unsafe character list contains the language of SAS characters:

- ' - single quote
- " - double quote
- ; - semi-colon
- & - ampersand

The unsafe character list can be changed via the `UNSAFE` option - but note that any custom list should include the default characters unless there is a very good reason not to.

The following example starts an Application Server that removes control characters before they are assigned to macro variables. Note that this example usage of the `UNSAFE` option simply adds the default unsafe character list, and so has no practical effect:

```
PROC APPSRV PORT=5001 UNSAFE="''";&" ;  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

The next example adds spaces and vowels to the list of unsafe characters:

```
PROC APPSRV PORT=5001 UNSAFE="''";& aeiou";  
    ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';  
    PROGLIBS demo;  
RUN;
```

WEBOUT_ENCODING

The `WEBOUT_ENCODING` option sets the encoding of the `_WEBOUT` fileref that an Application Server provides to the programs it executes. Program output written to `_WEBOUT` is returned via the Broker to the user.

This option is particularly important if an Application Server runs in EBCDIC environments such as z/OS. Most web browsers do not recognise EBCDIC encodings.

UTF-8 is often a good, practical choice, since this is understood by many browsers and web clients and supports the display of multi-byte characters.

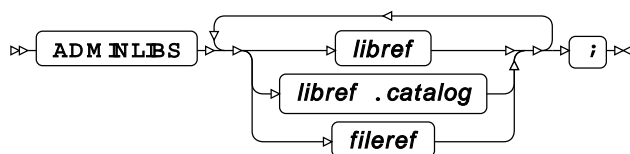
This option does not affect the `_GRPHOUT` fileref, which has no encoding, nor does it affect the automatic header generation system, which always emits headers in ASCII.

The following example sets the encoding of the `_WEBOUT` fileref to `latin1`:

```
PROC APPSRV PORT=5001 WEBOUT_ENCODING='latin1';
  ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';
  PROGLIBS demo;
  RUN;
```

ADMINLIBS

The `ADMINLIBS` statement of the `APPSRV` procedure specifies a list of libraries or directories from which programs can be executed by an Application Server if a valid password is provided via the `_ADMINPW` parameter. `ADMINLIBS` can be used to distinguish administrative programs from general programs in an application. The relevant libraries or directories must have been allocated using the `ALLOCATE` statement.



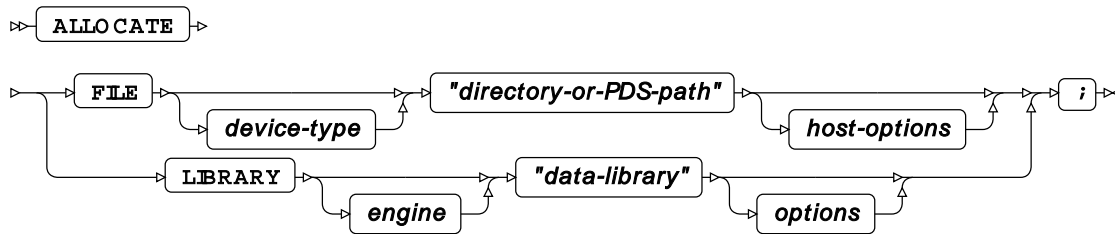
All `ADMINLIBS` arguments are the names of librefs, catalogs or filerefs.

This example program starts an Application Server that permits programs in the `C:\wpswebapp\wpsdemoapp\admin` directory to be executed if a valid `_ADMINPW` parameter is provided.

```
PROC APPSRV PORT=5001 ADMINPW="foo";
  ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";
  ALLOCATE FILE admin "C:\wpswebapp\wpsdemoapp\admin";
  PROGLIBS demo;
  ADMINLIBS admin;
  RUN;
```

ALLOCATE

The **ALLOCATE** statement of the APPSRV procedure defines libraries, directories and files as available for use by an Application Server. These definitions can be passed as parameters to other statements like **PROGLIBS** to enable further functionality.



ALLOCATE LIBRARY and **ALLOCATE FILE** are similar to the global statements **LIBNAME** and **FILENAME** in the language of SAS, but scope is restricted to the Application Server only.

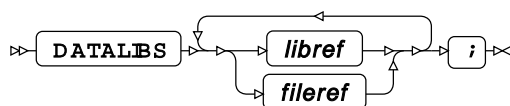
The example below allows programs stored as files in `C:\wpswebapp\wpsdemoapp\server` to be executed, passing an extra library reference to each program which maps to the `C:\wpswebapp\wpsdemoapp\data` directory:

```

PROC APPSRV PORT=5001 ADMINPW="foo";
ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";
ALLOCATE LIBRARY data "C:\wpswebapp\wpsdemoapp\data";
PROGLIBS demo;
DATALIBS data;
RUN;
  
```

DATALIBS

The **DATALIBS** statement of the APPSRV procedure defines a space-separated list of allocated library references that are passed to programs executed by an Application Server. This allows data to be stored globally between requests or permanently as required. This facility is particularly useful as the **APSWORK** library is cleared when the server is stopped and is therefore only suitable for temporary data used during program execution.



All **DATALIBS** arguments are the names of librefs or filerefs, which have to be allocated with the **ALLOCATE** statement.

Warning:

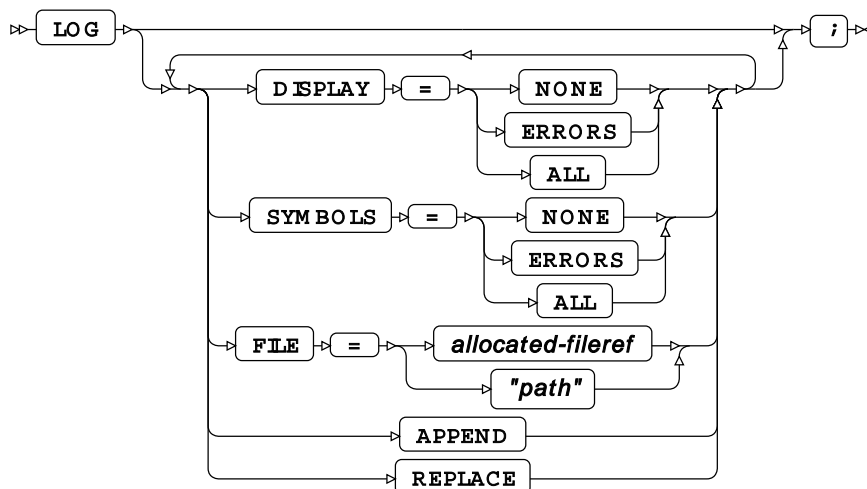
Creating a library using **DATALIBS** which is called **SAVE** may interfere with the **SAVE** library used by Application Server sessions. Therefore, this name should be avoided.

The following example starts an Application Server that passes three libraries to the programs it executes:

```
PROC APPSRV PORT=5001 ADMINPW="foo";
ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";
ALLOCATE LIBRARY seta "C:\wpswebapp\wpsdemoapp\seta";
ALLOCATE LIBRARY setb "C:\wpswebapp\wpsdemoapp\setb";
ALLOCATE LIBRARY setc "C:\wpswebapp\wpsdemoapp\setc";
PROGLIBS demo;
DATALIBS seta setb setc;
RUN;
```

LOG

The **LOG** statement of the APPSRV procedure is used to configure logging of the requests that an Application Server receives.



APPEND and REPLACE

The **APPEND** and **REPLACE** options of the **LOG** statement define how logs are opened. If **APPEND** is specified, then log output is appended to an existing log file. If **REPLACE** is specified, then a log file is always replaced. If neither is specified, then the behaviour is operating-system dependent. On z/OS the log file is always appended to, whereas on other operating systems, the log file is replaced if it is older than six days.

DISPLAY

The **DISPLAY** option of the **LOG** statement controls whether the Application Server records the execution log for each requested program.

If it is set to **NONE**, then the execution log is never included. If it is set to **ERRORS**, then the execution log is recorded if there is an error. If set to **ALL** then the execution log is always recorded. By default, it is set to **NONE**.

The example below configures an Application Server to include the execution log for every request:

```
PROC APPSRV PORT=5001;
  ALLOCATE FILE logfile "C:\wpswebapps\wpsdemoapp\appsrv-%Y-%m-%d.log";
  LOG FILE=logfile DISPLAY=ALL;
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
  PROGLIBS demo;
RUN;
```

FILE

The **FILE** option of the **LOG** statement is used to specify a fileref or a string containing a filename. The name may include patterns which are expanded as the Application Server writes the log. This is useful, for example, if you need to embed creation dates within the names of log files. The *Substitution Characters Table* [↗](#) (page 2283) details the formatting patterns that can be used.

In the example below, we specify a log file using a file path string:

```
PROC APPSRV PORT=5001;
  LOG FILE="C:\wpswebapps\wpsdemoapp\appsrv.log";
  ALLOCATE FILE demo 'C:\wpswebapp\wpsdemoapp\server';
  PROGLIBS demo;
RUN;
```

Here, we specify a file path using an allocated fileref:

```
PROC APPSRV PORT=5001;
  ALLOCATE FILE logfile "C:\wpswebapps\wpsdemoapp\appsrv.log";
  LOG FILE=logfile;
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
  PROGLIBS demo;
RUN;
```

Below, we configure logging so that it produces files with names such as `appsrv-2013-08-01.log` in the `wpsdemoapp` directory:

```
PROC APPSRV PORT=5001;
  ALLOCATE FILE logfile "C:\wpswebapps\wpsdemoapp\appsrv-%Y-%m-%d.log";
  LOG FILE=logfile;
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
  PROGLIBS demo;
RUN;
```

SYMBOLS

The **SYMBOLS** option of the **LOG** statement controls whether the Application Server logs input symbols/parameters from incoming requests. The values of symbols whose names begin with `_NOLOG_` have their values obfuscated in the log file.

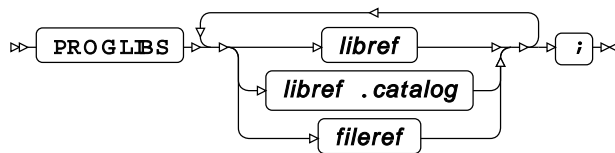
If set to **NONE**, no logging is performed. If set to **ERRORS** then symbols are logged if there is an error processing the request. Setting it to **ALL** logs symbols for all requests. By default, this value is set to **NONE**.

In the following example, an Application Server is configured to record all symbols for all requests:

```
PROC APPSRV PORT=5001;  
  ALLOCATE FILE logfile "C:\wpswebapps\wpsdemoapp\appsrv-%Y-%m-%d.log";  
  LOG FILE=logfile SYMBOLS=ALL;  
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";  
  PROGLIBS demo;  
RUN;
```

PROGLIBS

The `PROGLIBS` statement of the `APPSRV` procedure defines a space-separated list of allocated library, catalog or file references that contain programs which can be executed by an Application Server. Specifying the catalog name after the library name ensures that only programs in that catalog are accessible.



All `PROGLIBS` arguments are the names of `librefs`, `filerefs` or `catalogs`.

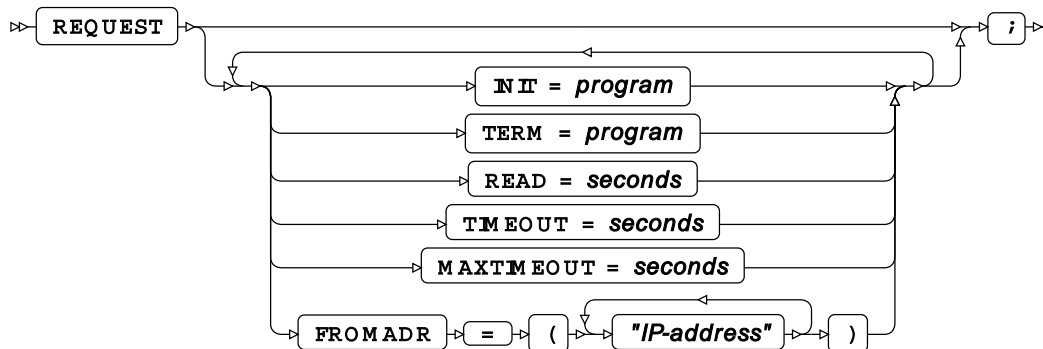
This example allows execution of:

- All programs in the server directory using the prefix `demo`
- All programs contained in any catalog in the `proglib` library
- All programs contained in the `catalog1` and `catalog2` catalogs of the `adlib` library

```
PROC APPSRV PORT=5001;  
ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";  
ALLOCATE LIBRARY proglib "C:\wpswebapp\wpsdemoapp\proglib";  
ALLOCATE LIBRARY adlib "C:\wpswebapp\wpsdemoapp\admin";  
PROGLIBS demo proglib adlib.catalog1 adlib.catalog2;  
RUN;
```


REQUEST

The `REQUEST` statement of the `APPSRV` procedure is used to configure aspects of Application Server request processing.



FROMADR

The `FROMADR` option of the `REQUEST` statement is used to enhance security by providing a white-list of IP addresses from which a Broker will accept requests. All other requests will be denied with an error message and the Application Server will stop processing the request. The `FROMADR` list is a space-separated list of strings contained in parentheses.

This example specifies that the Broker should only accept requests from the local host:

```

PROC APPSRV PORT=5001;
  REQUEST FROMADR= ('127.0.0.1');
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
  PROGLIBS demo;
RUN;

```

INIT

The `INIT` option of the `REQUEST` statement specifies the name of a program to execute before the program requested by the Application Server is executed. This is implemented via the use of the `%INCLUDE` macro in the language of SAS.

By default, no `INIT` program is executed.

This option does not affect special programs like `stop`, `ping`, `replay`, `status` or the program executed when there is an invalid session as specified by the `INVSESS` option of the `SESSION` statement.

The example below illustrates how to configure an Application Server to execute the `init.sas` program before any requested program:

```

PROC APPSRV PORT=5001;
  ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
  REQUEST INIT="demo.init.sas";
  PROGLIBS demo;
RUN;

```

MAXTIMEOUT

The **MAXTIMEOUT** option of the **REQUEST** statement specifies the maximum **TIMEOUT** value that can be set for a program. This value overrides the value of the **TIMEOUT** option if the **TIMEOUT** option is set to a value higher than the **MAXTIMEOUT** value.

MAXTIMEOUT should be set to a value greater than zero and less than or equal to **INT_MAX** which is the largest integer representable on the specific operating system concerned. By default, it is set to 900 seconds or 15 minutes.

READ

The **READ** option of the **REQUEST** statement specifies the maximum duration allowed for the reading of a request from a Broker in seconds. If a request cannot be read within this time, the server will respond with an HTTP message containing the status header *408 - Request Timed Out*.

READ must be set to a value greater than zero and less than or equal to **INT_MAX** which is the largest integer it is possible to represent on the specific operating system concerned. By default, **READ** is set to 30 seconds.

TERM

The **TERM** option of the **REQUEST** statement specifies the name of a program to execute after a program requested by the user. This is implemented via the **%INCLUDE** macro in the language of SAS.

By default, no program is executed.

This option does not affect special programs like `stop`, `ping`, `replay`, `status` or the program executed when there is an invalid session as specified by the **INVSESS** option of the **SESSION** statement.

This example shows an Application Server that executes the `term.sas` program after any requested program:

```
PROC APPSRV PORT=5001;  
ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";  
REQUEST TERM="demo.term.sas";  
PROGLIBS demo;  
RUN;
```

TIMEOUT

The **TIMEOUT** option of the **REQUEST** statement specifies the maximum duration allowed for the execution of a request including extensions prepended and appended by the **INIT** and **TERM** options of the **SESSION** or **REQUEST** statements.

When the duration of a request has exceeded the timeout value, the program is cancelled. It is possible that this will leave an Application Server in an unexpected state.

A program can increase the time it is allowed to execute for using the **APPSRV_SET()** function. The maximum timeout which a program can set for a request is limited by the **MAXTIMEOUT** option of the **REQUEST** statement.

This value is overridden by the value set in `MAXTIMEOUT` if the `TIMEOUT` value is set to a value higher than `MAXTIMEOUT`.

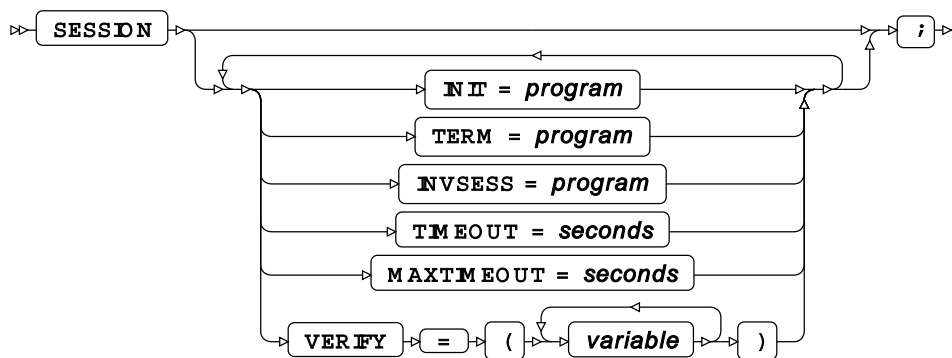
The value set in `TIMEOUT` must be greater than zero and less than or equal to that set in `MAXTIMEOUT`. The default value for `TIMEOUT` is 300 seconds.

The following program starts an Application Server which allows programs to execute for at most 120 seconds:

```
PROC APPSRV PORT=5001;
ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";
REQUEST TIMEOUT=120;
PROGLIBS demo;
RUN;
```

SESSION

The `SESSION` statement is used to configure an Application Server's session functionality. The options listed are not mandatory.



INIT

The `INIT` option of the `SESSION` statement specifies the name of a program to be executed when a session is created. The program defined by the `INIT` option can access:

- The session `SAVE` library
- Libraries created through the `DATALIBS` statement
- The `_SESSIONID` macro variable
- A unique `WORK` library that is terminated at the end of the program

The program cannot access:

- The `_WEBOUT` or `_GRPHOUT` filerefs
- User-provided parameters or macro variables created by the program that launched the session
- The `APSWORK` library

By default, no program is executed.

This example launches an Application Server which executes the `session_init.sas` program whenever sessions are created:

```
PROC APPSRV PORT=5001;  
ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";  
SESSION INIT="demo.session_init.sas";  
PROGLIBS demo;  
RUN;
```

INVSESS

The `INVSESS` option of the `SESSION` statement specifies the name of a program when a request is received with an invalid `_SESSIONID` parameter. This can be used to override the default response from an application server, which in its raw form, resembles the image below:



The specified `INVSESS` program can access the `APSWORK` and `DATALIBS` libraries and inputs through macro variables just like a regular program. However, the `_PROGRAM` macro variable becomes the value of `INVSESS` and a macro variable called `_USERPROGRAM` is created and holds the name of the program that was attempted to be executed.

In this example, an Application Server executes the `session_invalidsession.sas` program when an invalid `_SESSIONID` parameter is received:

```
PROC APPSRV PORT=5001;  
ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";  
SESSION INVSESS="demo.session_invalidsession.sas";  
PROGLIBS demo;  
RUN;
```

MAXTIMEOUT

The `MAXTIMEOUT` option of the `SESSION` statement specifies the maximum duration of sessions in seconds. It overrides the value set by the `TIMEOUT` option if the `TIMEOUT` value is set to a higher value than `MAXTIMEOUT`.

`MAXTIMEOUT` should be set to a value greater than zero and less than or equal to `INT_MAX` which is the maximum value an integer can take in the language of SAS on that specific platform. By default, `MAXTIMEOUT` is set to `INT_MAX`.

TERM

The `TERM` option of the `SESSION` statement specifies the name of a program to be executed when a session terminates - either explicitly or if the timeout period is exceeded. This program can access:

- The session `SAVE` library
- Libraries created through the `DATALIBS` statement

- The `_SESSIONID` macro variable
- A unique `WORK` library that is terminated at the end of the program

The program cannot access:

- The `_WEBOUT` or `_GRPHOUT` filerefs
- User-provided parameters or macro variables created by the program that launched the session
- The `APSWORK` library

By default, no `TERM` program will be executed, and the program name is not validated until the first session is removed.

This example executes the `session_term.sas` program when sessions end:

```
PROC APPSRV PORT=5001;  
  ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";  
  SESSION TERM="demo.session_term.sas";  
  PROGLIBS demo;  
RUN;
```

TIMEOUT

The `TIMEOUT` option of the `SESSION` statement specifies the maximum duration of sessions in seconds, and can only be used once per `SESSION` statement. The value is overridden by the value of `MAXTIMEOUT` if it has been set higher than `MAXTIMEOUT`.

`TIMEOUT` must be set to a value greater than zero and less than or equal to the value set as `MAXTIMEOUT`. The default is 900 seconds - 15 minutes.

VERIFY

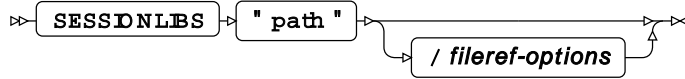
The `VERIFY` option of the `SESSION` statement defines a space-separated list of parameter names. When a session is created, the values of these parameters are recorded, so that, when an attempt is made to reconnect to the same session, these values can be verified against the parameter values of the new request to ensure that they are the same.

This example instructs the Application Server to verify the values of `FOO`, `BAR` and `BAZ`:

```
PROC APPSRV PORT=5001;  
  ALLOCATE FILE demo "C:\wpswebapp\wpsdemoapp\server";  
  PROGLIBS demo;  
  SESSION VERIFY=(FOO BAR BAZ);  
RUN;
```

SESSIONLIBS

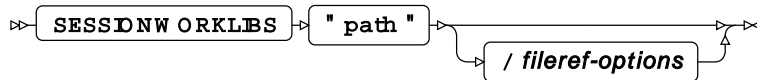
This statement controls the location of the `SAVE` library for each request.



The *Substitution Characters Table* [↗](#) (page 2283) can be used with this option. For an example of its use, refer to the `FILE` option of the `LOG` statement.

SESSIONWORKLIBS

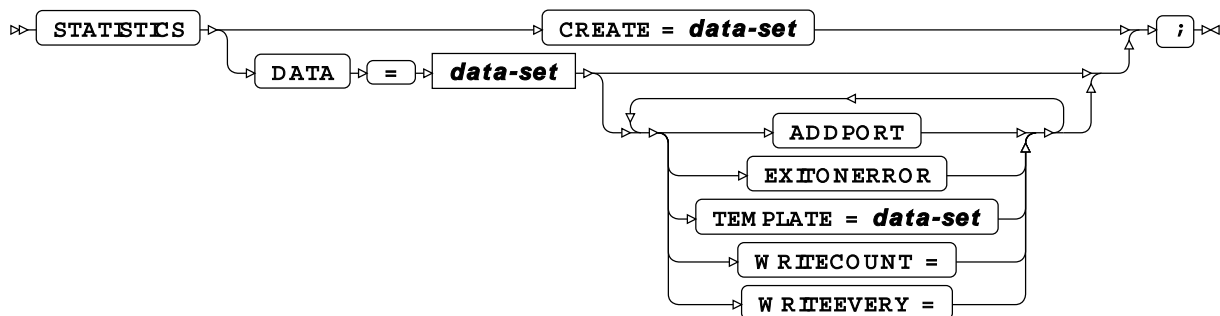
This statement controls the location of the `WORK` library for the session `INIT` and `TERM` programs that are executed when the session is created or destroyed.



The *Substitution Characters Table* [↗](#) (page 2283) can be used with this option. For an example of its use, refer to the `FILE` option of the `LOG` statement.

STATISTICS

The `STATISTICS` statement of the `APPSRV` procedure instructs an Application Server to generate a dataset that records requests received. This dataset can be accessed via special macro variables by programs executed by the Application Server.



ⁱ See *Dataset* [↗](#) (page 16).

ADDPOR

The **ADDPOR** keyword option of the **STATISTICS** statement determines whether the port number of the Application Server should be appended to the name specified in the **DATA** option to form the name of the dataset used to log statistics.

CREATE

The **CREATE** option of the **STATISTICS** statement is used to create a blank dataset with columns set to the defaults for Application Server statistics recording. The option prevents **PROC APPSRV** from binding to a socket and acting as a normal socket. The only resulting action is the creation of a blank dataset with the required columns for recording statistics.

The default columns of the statistics dataset are described in the table below:

Column	Column type	Description
Obstype	Character	Type of request
Okay	Character	1 if request ran OK, else 0
Duplex	Character	
Http	Character	
Program	Character	Name of program executed by the Application Server
Peeraddr	Character	
Hostname	Character	
Username	Character	
Entry	Character	Name of the item that was replayed if the replay program was used
SessionId	Character	ID of the session, if there was one
Service	Character	Name of the service the Application Server was part of
Starttime	Numeric	Date-time the request began
Runtime	Numeric	Number of seconds taken to process the request
Port	Numeric	Port number of the Application Server
Bytesin	Numeric	Size of the request in bytes
Bytesout	Numeric	Size of the response in bytes
Cputime	Numeric	Milliseconds of CPU time spent handling the request

DATA

The `DATA` option of the `STATISTICS` statement specifies the dataset used to record statistics and must be in a library that is allocated by the Application Server.

If the specified dataset does not exist already, then it is created and given a set of columns according to those specified in the `TEMPLATE` option.

In the following example, the dataset `requests` in the `stats` library is assigned to statistics-recording duties:

```
PROC APPSRV PORT=5001;  
ALLOCATE LIBRARY stats "C:\wpswebapps\wpsdemoapp\stats";  
ALLOCATE FILE demo "C:\wpswebapps\wpsdemoapp\server";  
PROGLIBS demo;  
STATISTICS DATA=stats.requests;  
RUN;
```

STARTTIMEMILLIS

The `STARTTIMEMILLIS` option of the `STATISTICS` statement specifies that the `starttime` will be recorded in milliseconds, rather than the default seconds.

TEMPLATE

The `TEMPLATE` option of the `STATISTICS` statement specifies the name of a dataset whose variables determine which statistics are recorded by the Application Server in the statistics dataset. These must be a subset of the full set as described in the `CREATE` option. If the `TEMPLATE` option is used, then for a variable to be included in the statistics dataset, there must be a column with the same name and type in the template dataset. If the `TEMPLATE` option is not used, then all statistics variables will be captured.

WRITECOUNT

The `WRITECOUNT` option of the `STATISTICS` statement specifies the number of requests that must be received before statistics are flushed from memory into the dataset.

This value should be greater than or equal to zero and less than or equal to `INT_MAX` which is the largest integer representable on the specific operating system in use. By default, it is set to zero.

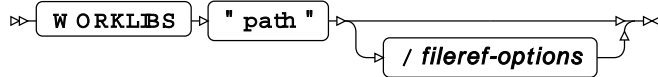
WRITEEVERY

The `WRITEEVERY` option of the `STATISTICS` statement determines the number of minutes after which statistics are flushed from memory into the dataset.

The value of this option should be greater than zero and less than or equal to `INT_MAX` which is the largest integer representable on the specific operating system in use. By default it is set to 5 minutes.

WORKLIBS

This statement controls the location of the `WORK` library for each request.



The *Substitution Characters Table* [↗](#) (page 2283) can be used with this option. For an example of its use, refer to the `FILE` option of the `LOG` statement.

Substitution Characters Table

This table details the formatting patterns that can be included when specifying filerefs or strings containing filenames.

The patterns are expanded as the Application Server writes the log. This is useful, for example, if you need to embed creation dates within the names of log files.

Format	Description	Range
%a	Day of week	Sun-Sat
%b	Month	Jan-Dec
%d	Number of day in month	01-31
%H	Hour	00-23
%m	Month	01-12
%w	Day of week	1-Sunday to 7-Saturday
%Y	Full year - e.g. 2015	
%y	Two-digit year	00-99
%p	Port number of Application Server	
%n	Hostname/node name	
%r	Request number	00001-99999
%s	Session number	

CATALOG procedure

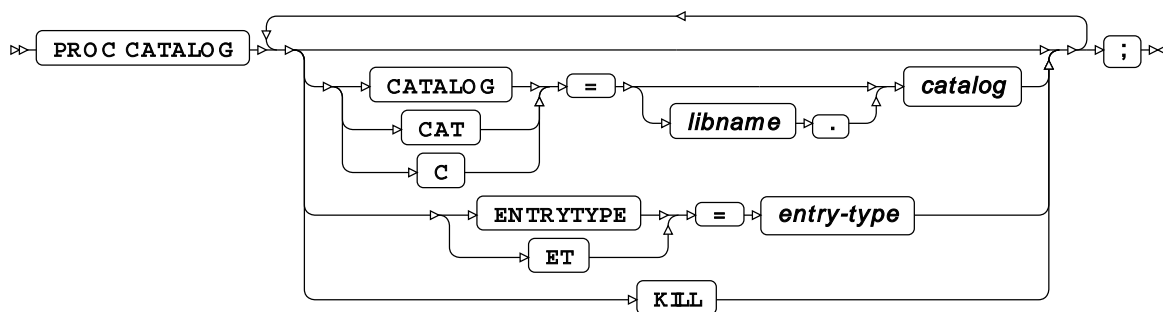
Supported statements

- `PROC CATALOG` [↗](#) (page 2284)

- [CHANGE](#) (page 2284)
- [CONTENTS](#) (page 2285)
- [COPY](#) (page 2285)
- [DELETE](#) (page 2285)
- [EXCHANGE](#) (page 2286)
- [EXCLUDE](#) (page 2286)
- [MODIFY](#) (page 2286)
- [SAVE](#) (page 2287)
- [SELECT](#) (page 2287)

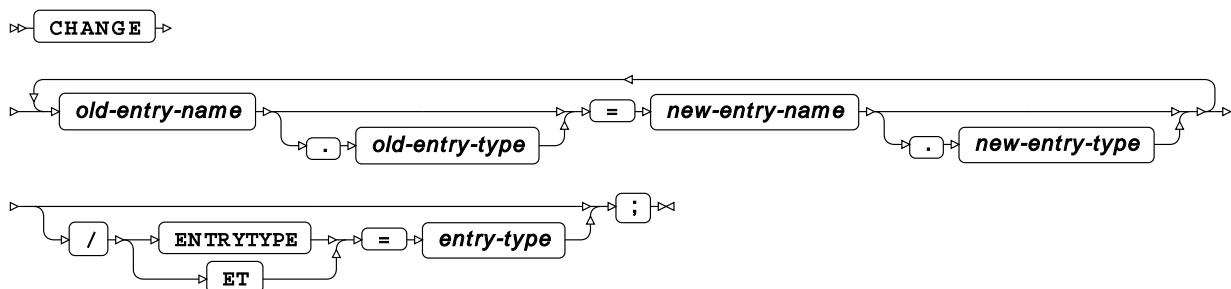
PROC CATALOG

Manages catalog files in libraries.



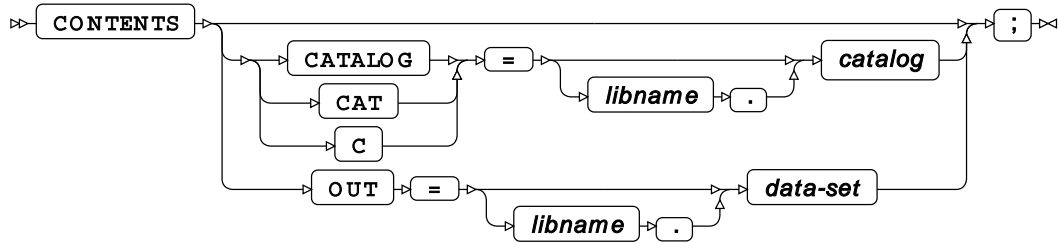
CHANGE

Renames catalog entries.



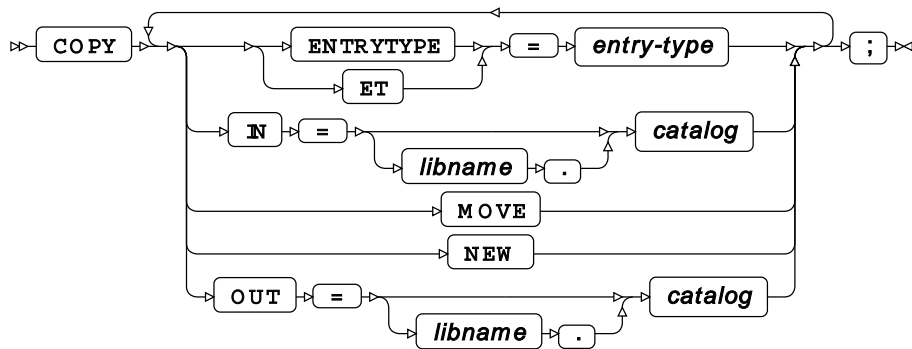
CONTENTS

Lists contents of a catalog.



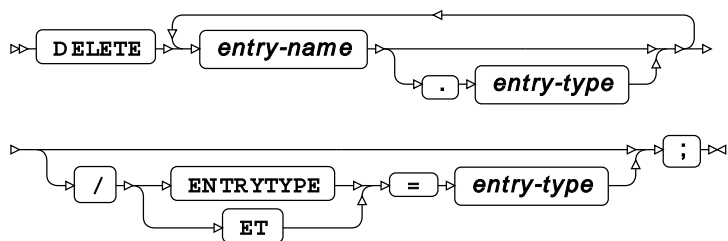
COPY

Copies all or part of a catalog to another catalog.



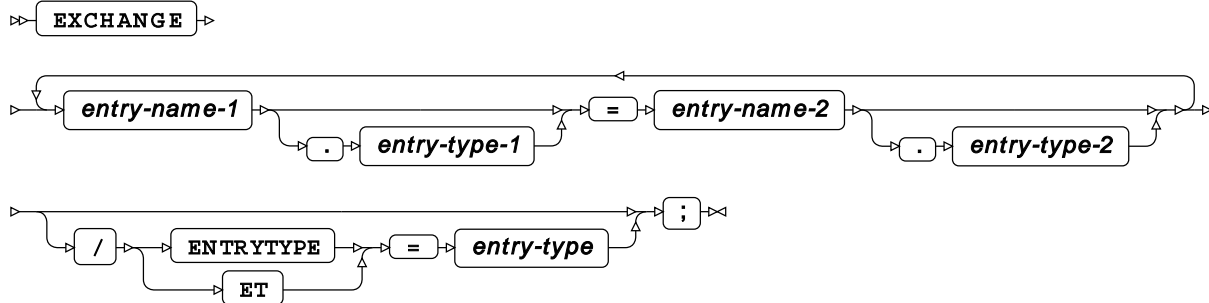
DELETE

Deletes one or more catalog entries.



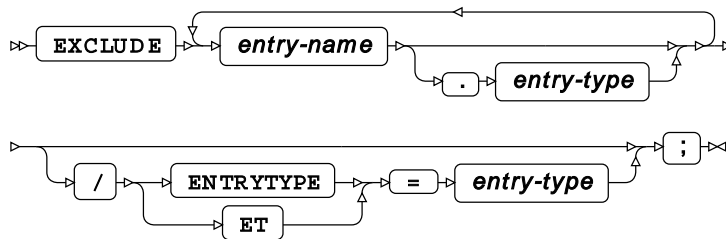
EXCHANGE

Exchanges entries within a catalog.



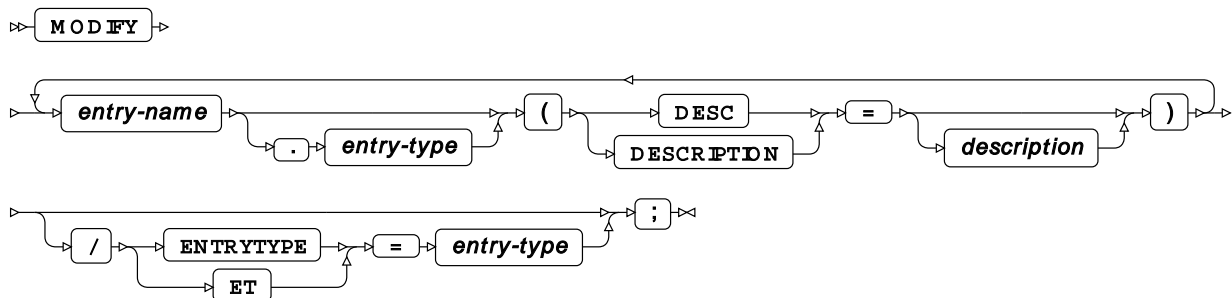
EXCLUDE

Specifies catalog entries to be excluded from other catalog operations.



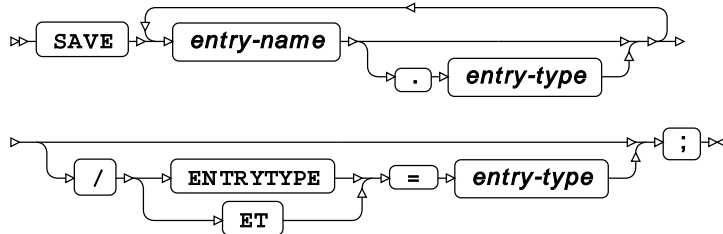
MODIFY

Modifies a catalog entry.



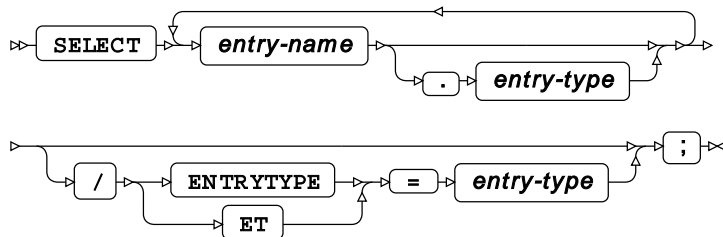
SAVE

Saves the specified catalog entries.



SELECT

Specifies catalog entries to be used in other catalog operations.



CDISC procedure

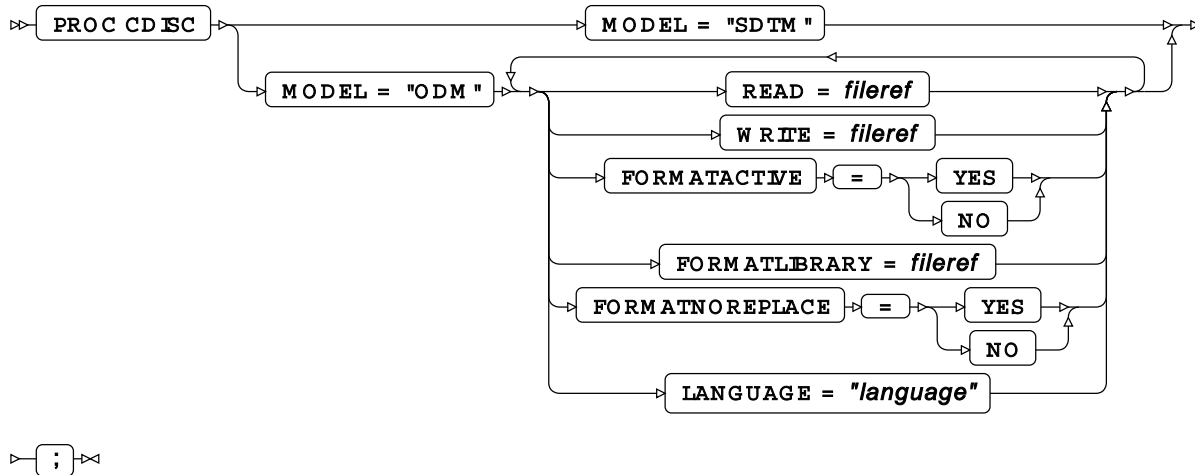
Supported statements

- *PROC CDISC* [↗](#) (page 2288)
- *ODM* [↗](#) (page 2289)
- *STUDY* [↗](#) (page 2290)
- *GLOBALVARIABLES* [↗](#) (page 2290)
- *USER* [↗](#) (page 2290)
- *LOCATION* [↗](#) (page 2290)
- *SIGNATURE* [↗](#) (page 2290)
- *CLINICALDATA* [↗](#) (page 2291)
- *CONTENTS* [↗](#) (page 2291)
- *DATASETS* [↗](#) (page 2291)
- *SDTM* [↗](#) (page 2291)

- [DOMAINDATA](#) (page 2292)

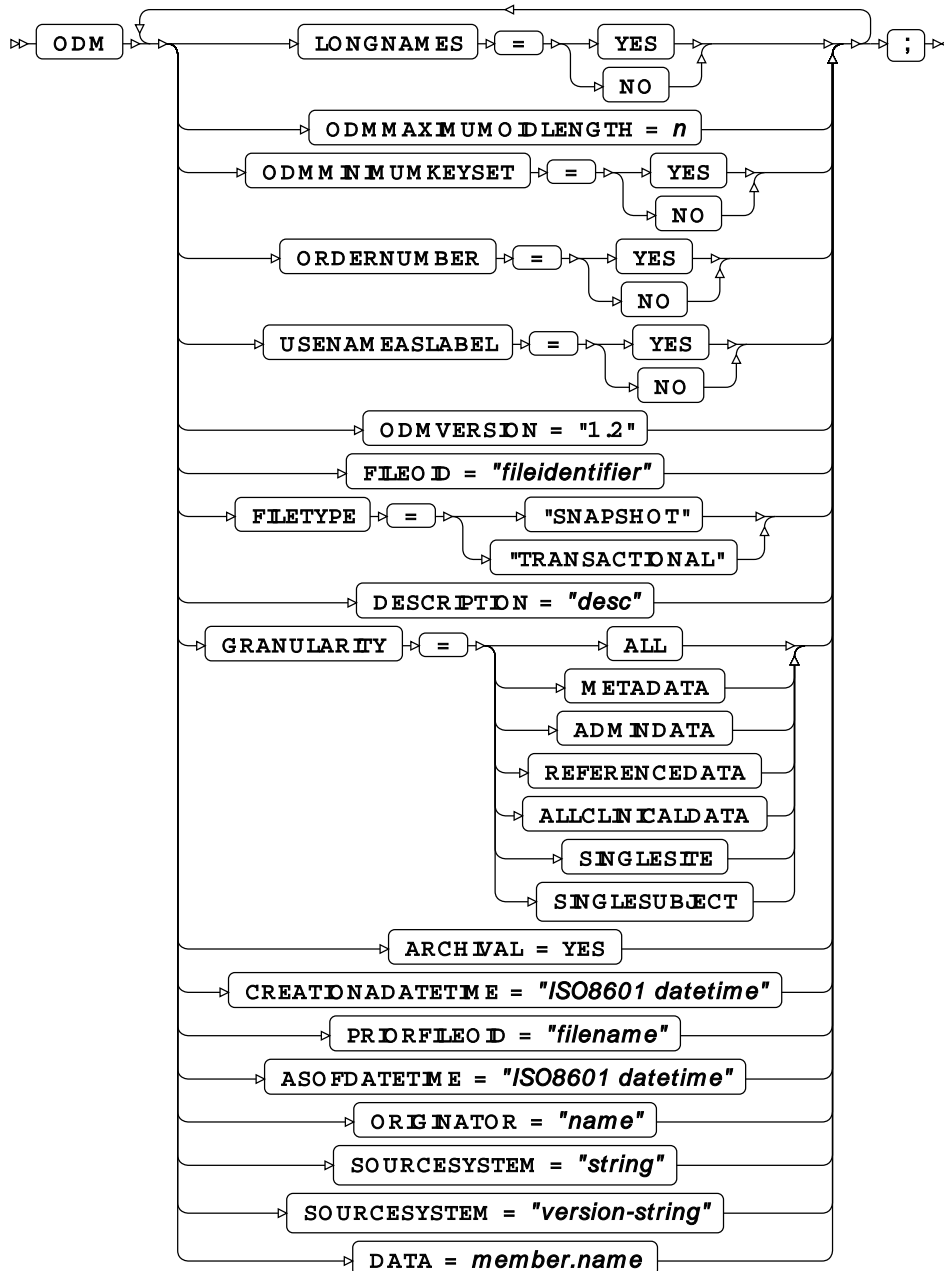
PROC CDISC

Reads and writes CDISC XML files, or validates SDTM datasets.



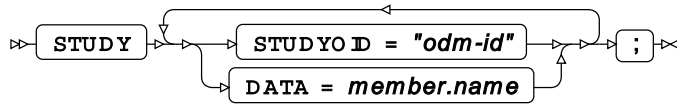
ODM

Defines various settings for ODM files imported or exported using PROC CDISC MODEL= "ODM".



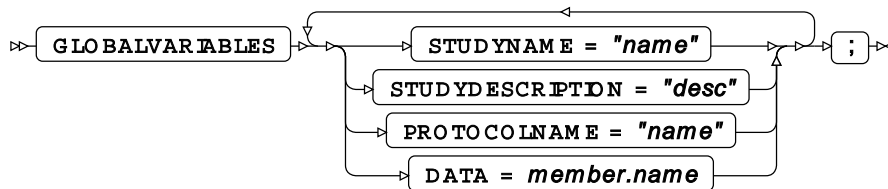
STUDY

Exports ODM by specifying the datasets containing the STUDYOID.



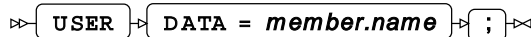
GLOBALVARIABLES

Specifies global variables when exporting.



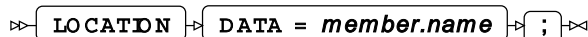
USER

Specifies the username when exporting.



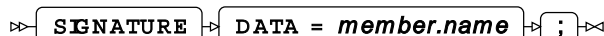
LOCATION

Specifies the specific location when exporting a dataset.



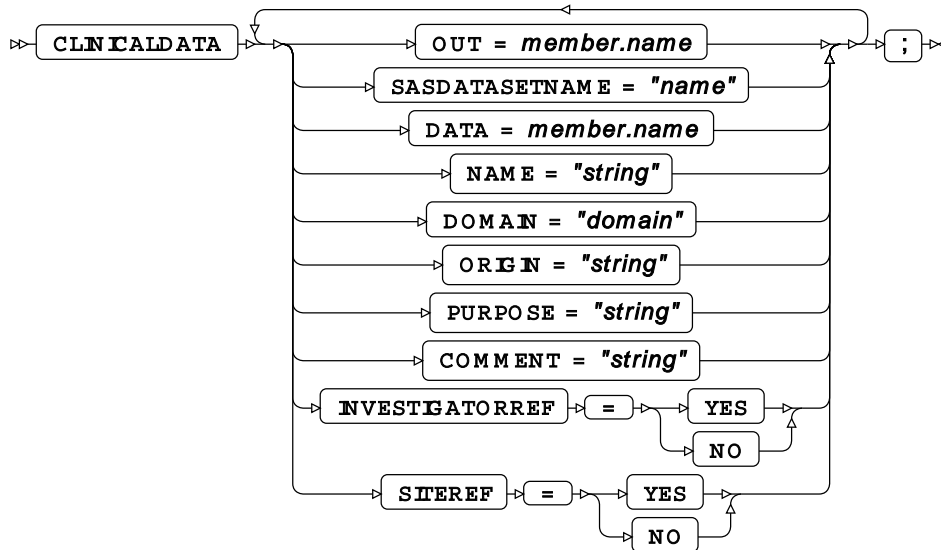
SIGNATURE

Specifies the identifier of the person entering the record.



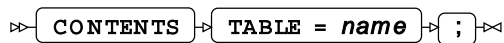
CLINICALDATA

Specifies the dataset to write to on import. Specifies the dataset to read from on export.



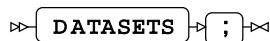
CONTENTS

Prints out the contents of the named dataset.



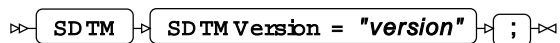
DATASETS

Reports the datasets in the imported file.



SDTM

Validates the contents of a dataset or XML file written in CDISC by checking fields and against the SDTM standard.



DOMAINDATA

Specifies the datasets to validate.

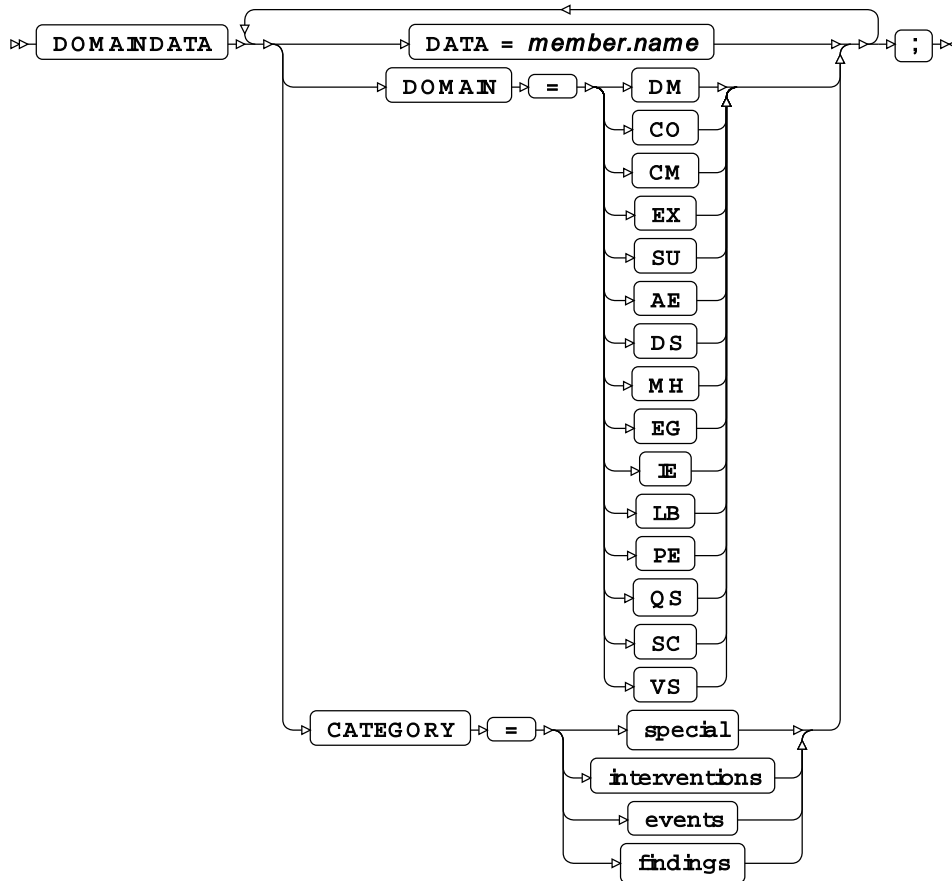


CHART procedure

Supported statements

- *PROC CHART* [↗](#) (page 2293)
- *HBAR* [↗](#) (page 2293)
- *VBAR* [↗](#) (page 2295)
- *BY* [↗](#) (page 2296)
- *WHERE* [↗](#) (page 2296)

PROC CHART

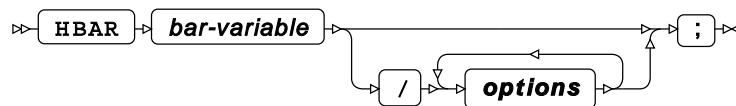
Generates a text-based bar chart from a dataset.



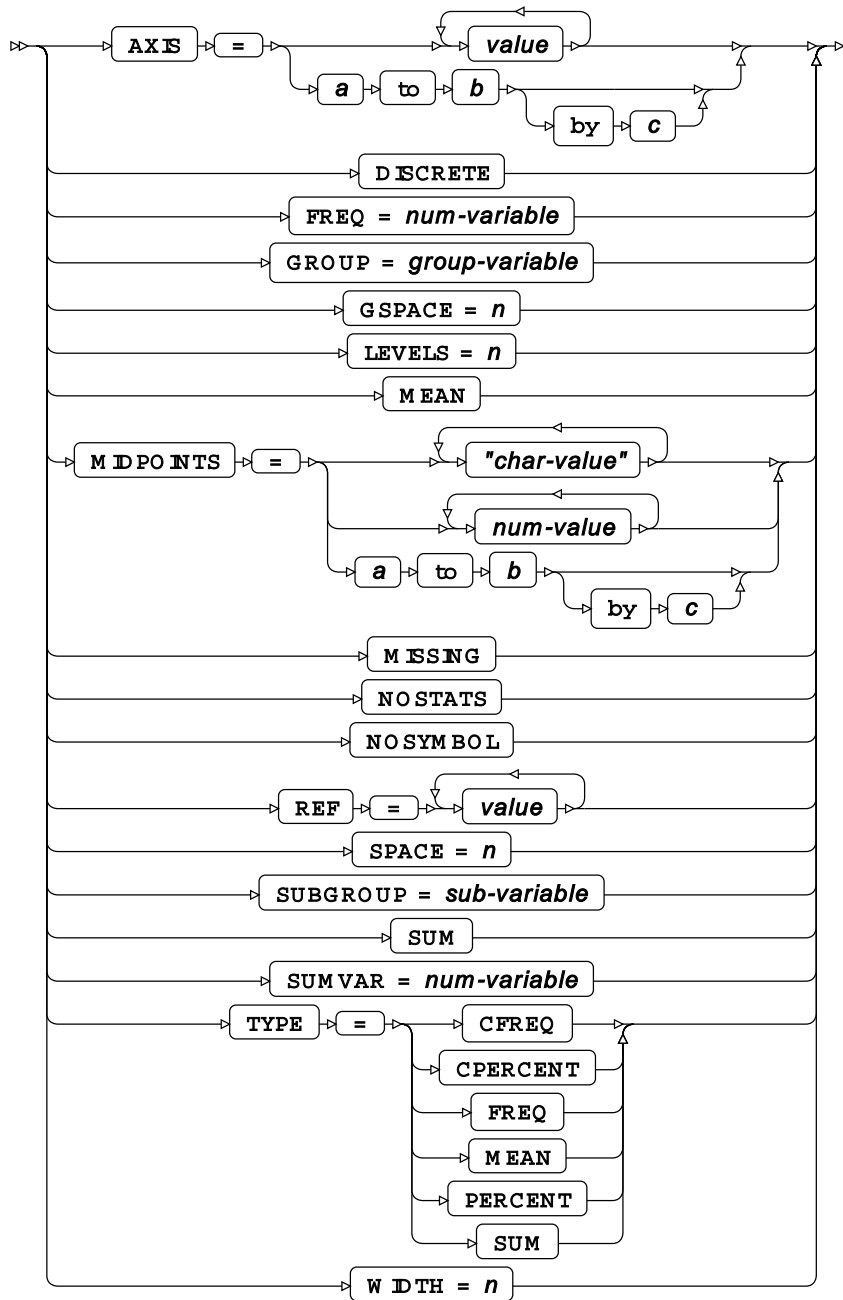
ⁱ See *Input dataset* [↗](#) (page 16).

HBAR

Creates a chart with horizontal bars, values increment from left to right.

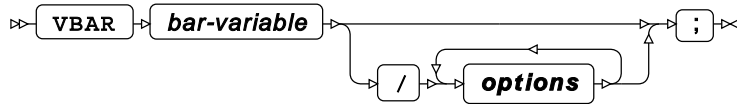


options

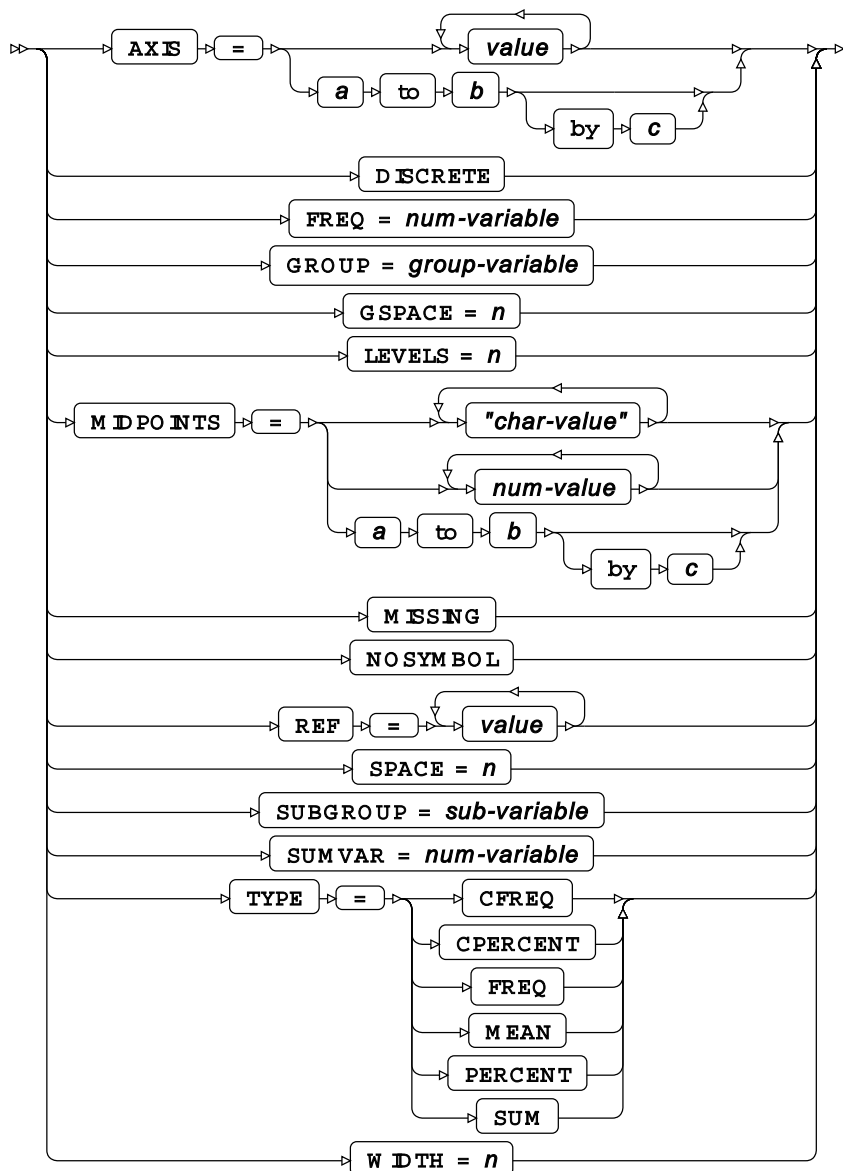


VBAR

Creates a chart with vertical bars, values increment from bottom to top.

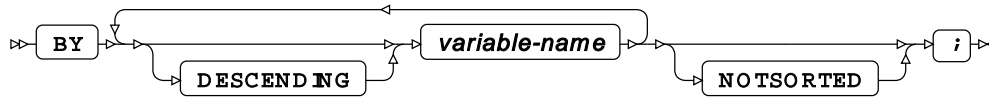


options



BY

Groups the observations in a dataset using one or more specified variables.



WHERE

Restricts the observations to be processed.



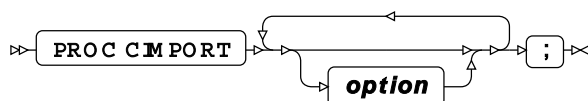
CIMPORT procedure

Supported statements

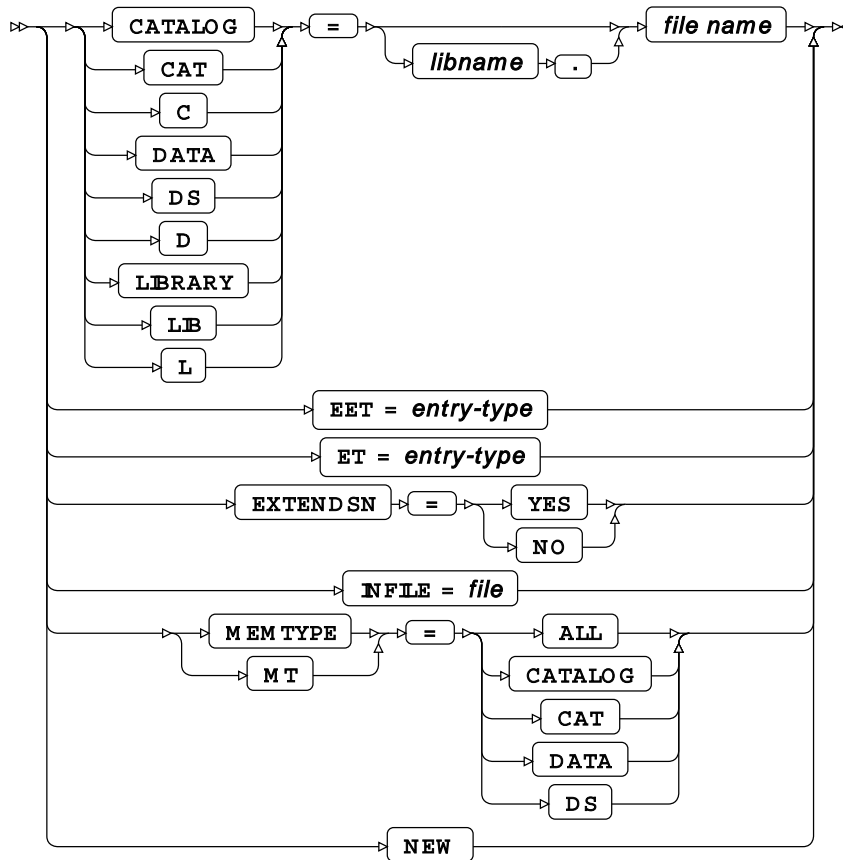
- *PROC CIMPORT* [↗](#) (page 2296)
- *EXCLUDE* [↗](#) (page 2297)
- *SELECT* [↗](#) (page 2298)

PROC CIMPORT

Imports a transport file containing catalog entries and metadata created using the **CPORT** procedure.

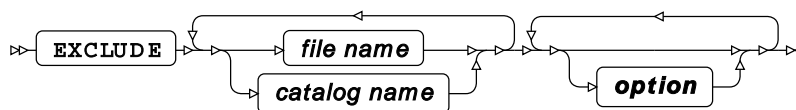


option

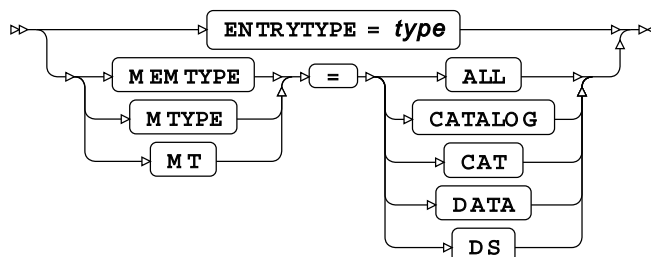


EXCLUDE

Includes all members except those listed.

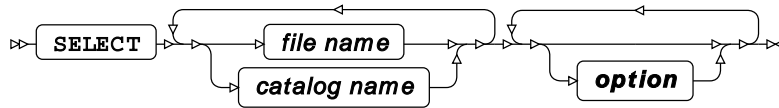


option

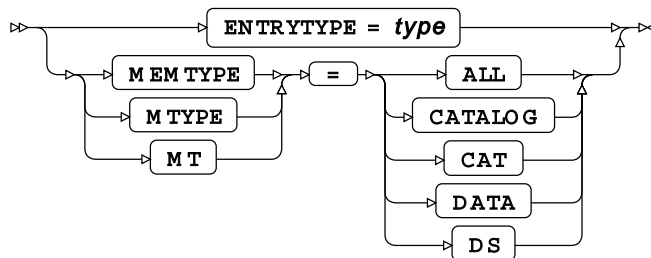


SELECT

Includes only the specified members.



option



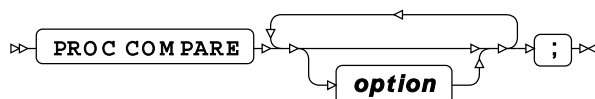
COMPARE procedure

Supported statements

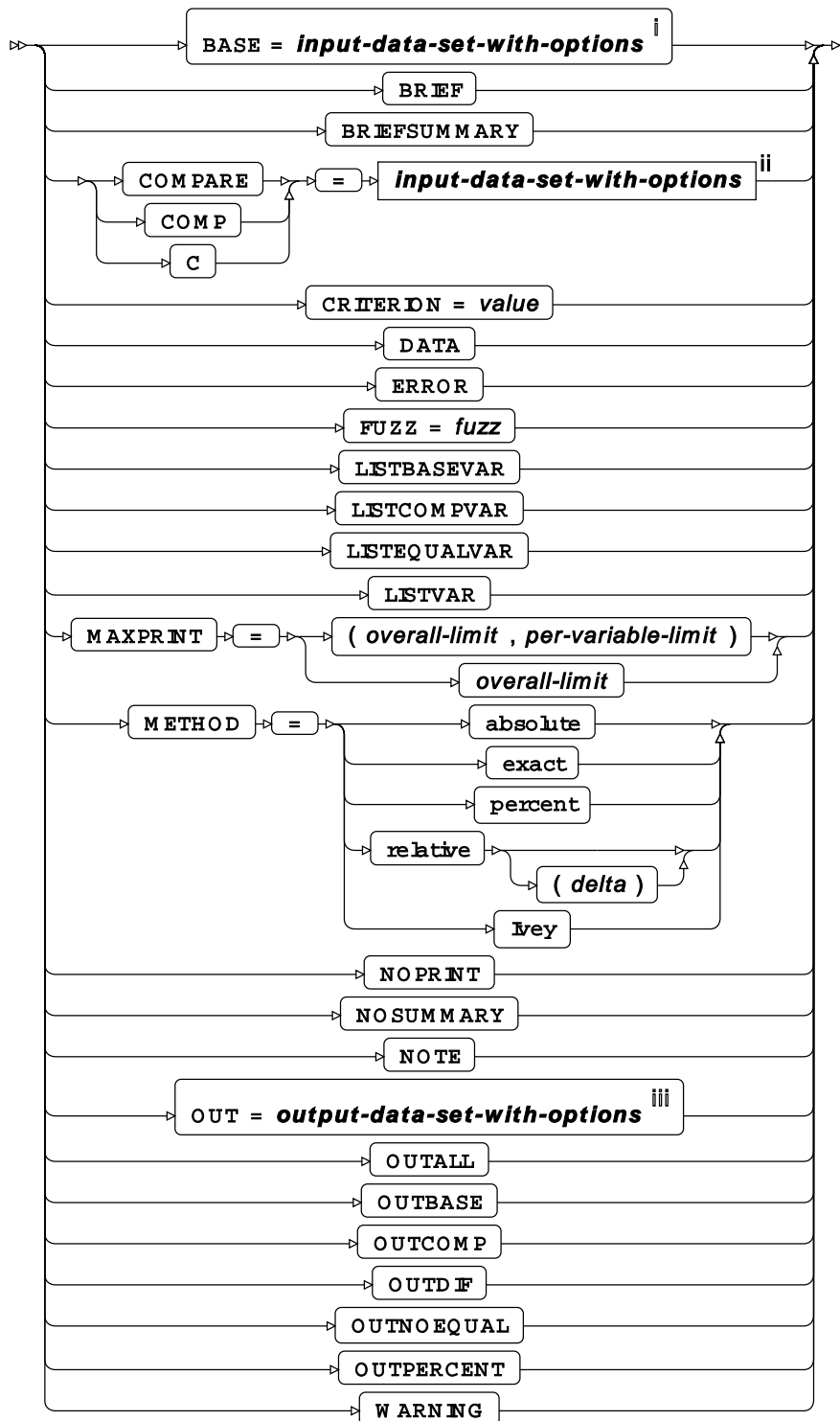
- *PROC COMPARE* [↗](#) (page 2298)
- *BY* [↗](#) (page 2300)
- *EXCLUDEVAR* [↗](#) (page 2300)
- *ID* [↗](#) (page 2300)
- *VAR* [↗](#) (page 2300)
- *WITH* [↗](#) (page 2301)
- *WHERE* [↗](#) (page 2301)

PROC COMPARE

Compares the metadata and data of two datasets.



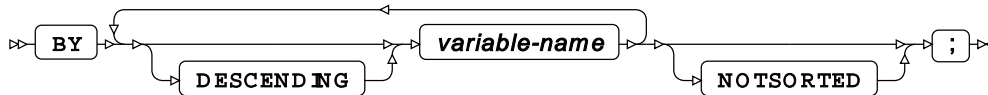
option

ⁱ See *Input dataset* [↗](#) (page 16).ⁱⁱ See *Input dataset* [↗](#) (page 16).

iii See *Output dataset* [↗](#) (page 16).

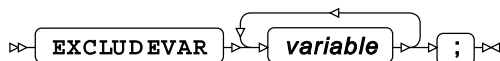
BY

Groups the observations in a dataset using one or more specified variables, and compares the grouped data.



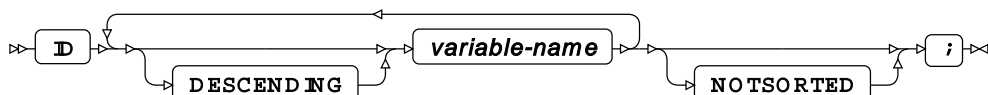
EXCLUDEVAR

Specifies variables to be omitted from the comparison of two datasets.



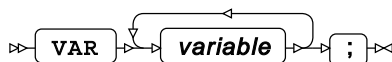
ID

Identifies the relevant observations in the output by using one or more specified variable names.



VAR

Compares only the listed variables, where the variables appear in both datasets.



WITH

Compares variables that are not identically named.



WHERE

Restricts the observations to be processed.



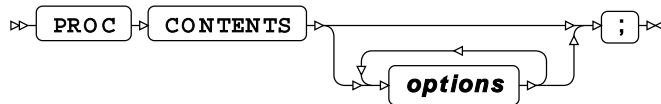
CONTENTS procedure

Supported statements

- *PROC CONTENTS* [↗](#) (page 2301)

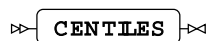
PROC CONTENTS

Describes one or more datasets in a library.



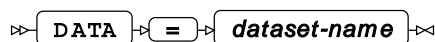
Options

CENTILES



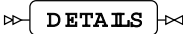
Type: Keyword

DATA



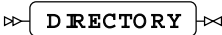
Type: String

DETAILS



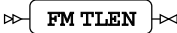
Type: Keyword

DIRECTORY



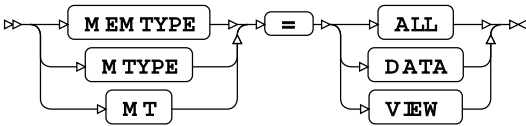
Type: Keyword

FMTLEN



Type: Keyword

MEMTYPE



ALL

DATA

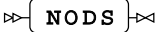
VIEW

NODETAILS

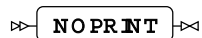


Type: Keyword

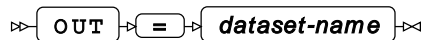
NODS



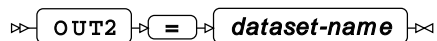
Type: Keyword

NOPRINT

```
»» NOPRINT ««
```

Type: Keyword**OUT**

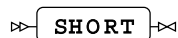
```
»» OUT = dataset-name ««
```

Type: String**OUT2**

```
»» OUT2 = dataset-name ««
```

Type: String**PRINT**

```
»» PRINT ««
```

Type: Keyword**SHORT**

```
»» SHORT ««
```

Type: Keyword**VARNUM**

```
»» VARNUM ««
```

Type: Keyword

COPY procedure

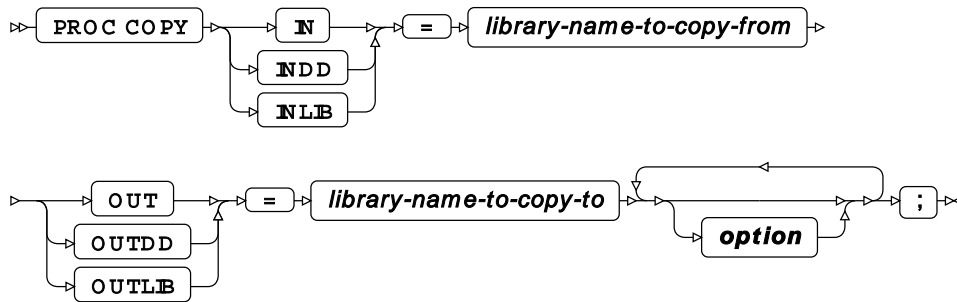
Supported statements

- `PROC COPY` [↗](#) (page 2304)
- `EXCLUDE` [↗](#) (page 2304)

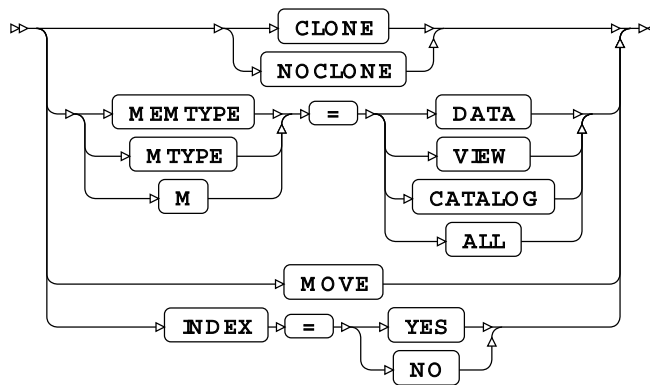
- [SELECT](#) (page 2305)

PROC COPY

Copies one or more datasets in a library to a different library.

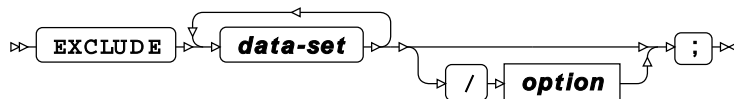


option

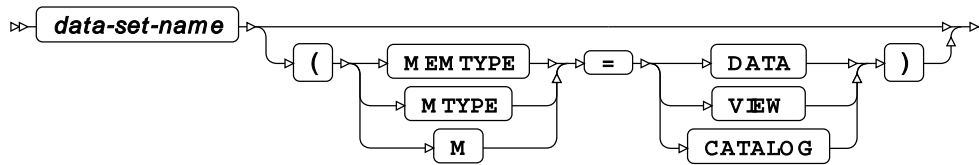


EXCLUDE

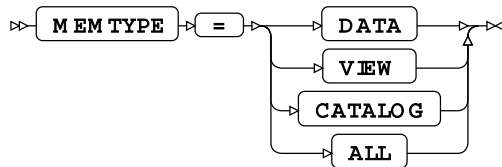
Copies all datasets except those listed to the new library.



data-set

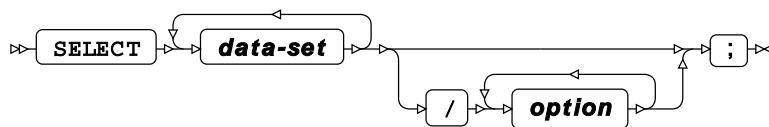


option

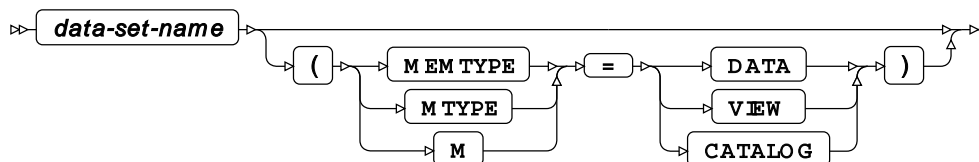


SELECT

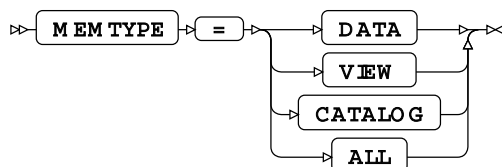
Copies only the datasets listed to the new library.



data-set



option



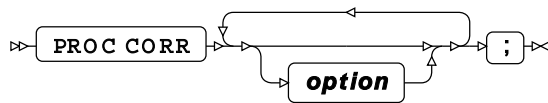
CORR procedure

Supported statements

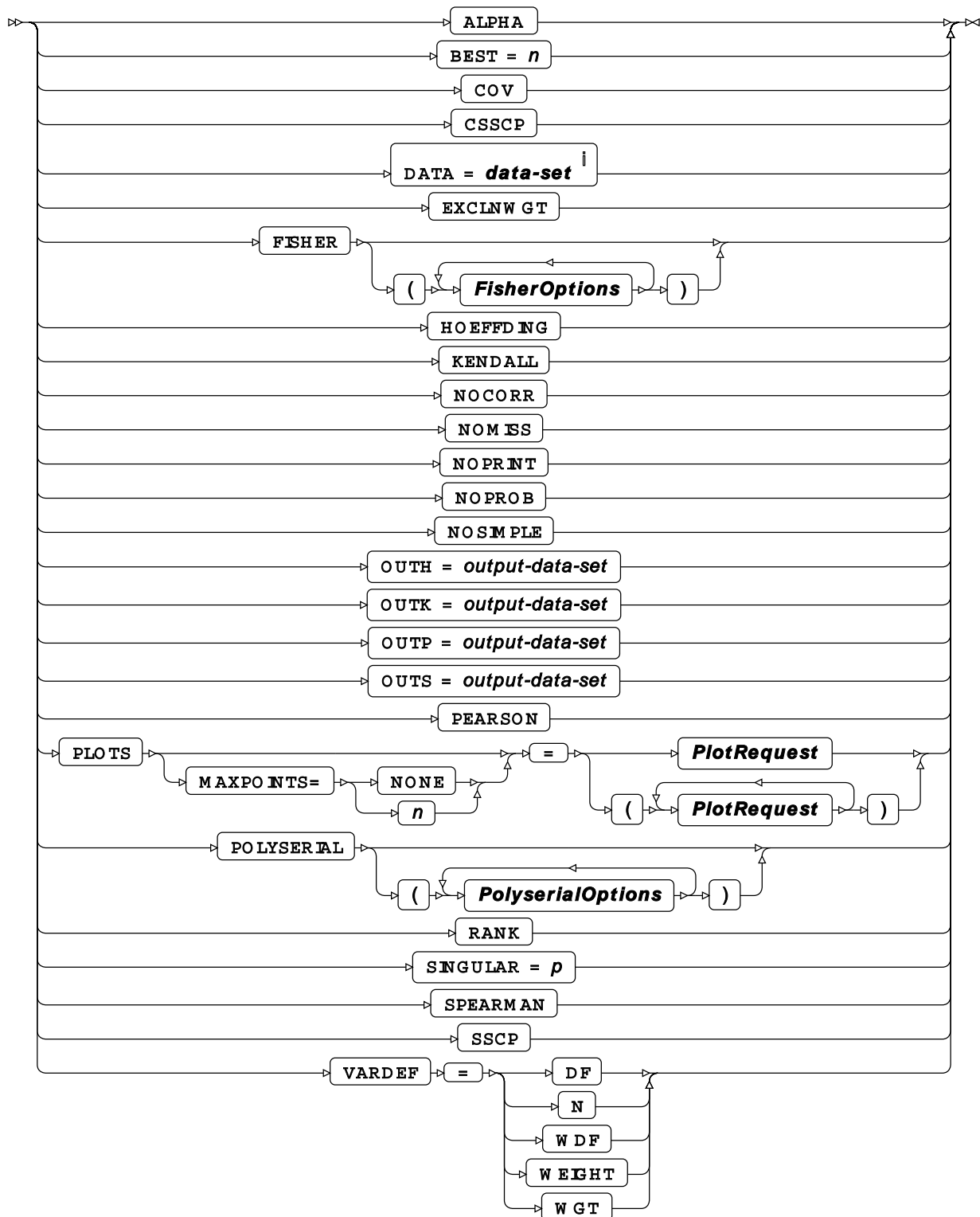
- *PROC CORR* [↗](#) (page 2306)
- *BY* [↗](#) (page 2309)
- *FREQ* [↗](#) (page 2309)
- *PARTIAL* [↗](#) (page 2309)
- *VAR* [↗](#) (page 2310)
- *WEIGHT* [↗](#) (page 2310)
- *WHERE* [↗](#) (page 2310)
- *WITH* [↗](#) (page 2310)

PROC CORR

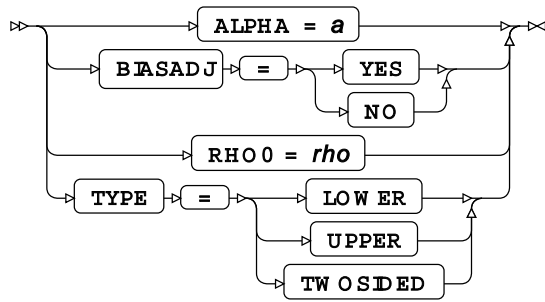
Performs correlation analysis against numeric variables.



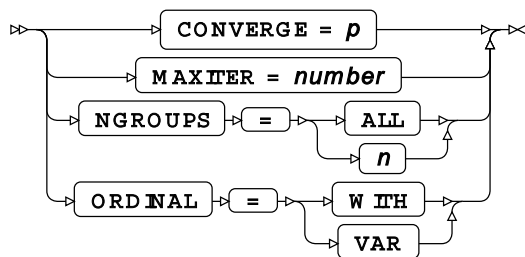
option

ⁱ See [Input dataset](#) (page 16).

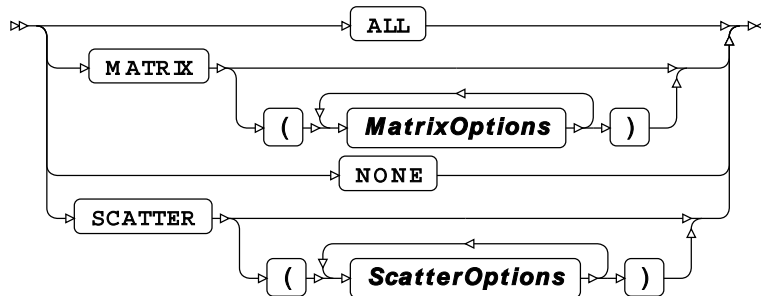
FisherOptions



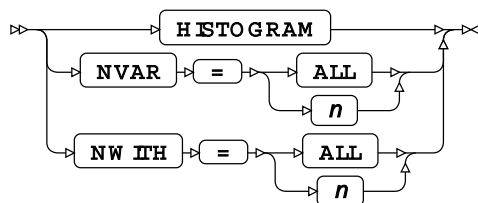
PolyserialOptions



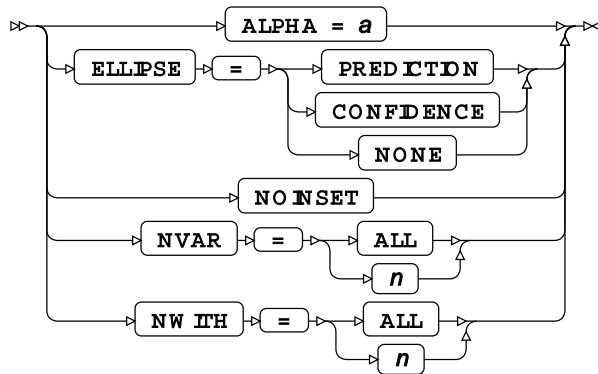
PlotRequest



MatrixOptions

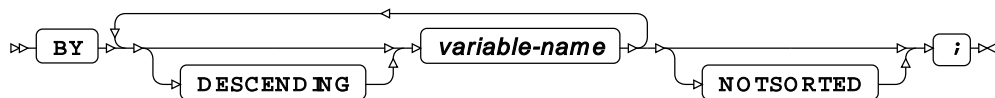


ScatterOptions



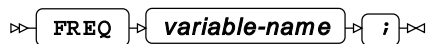
BY

Groups the observations in a dataset using one or more specified variables.



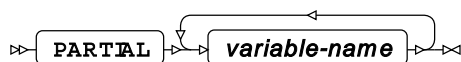
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



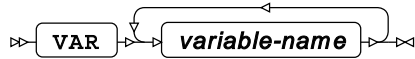
PARTIAL

Calculates partial correlations when the effect of the given variables are allowed for.



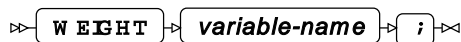
VAR

Correlates only the specified variables.



WEIGHT

Specifies a variable giving the weight associated with each observation.



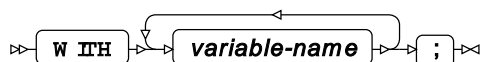
WHERE

Restricts the observations to be processed.



WITH

Calculates the correlation between specified pairs of variables.



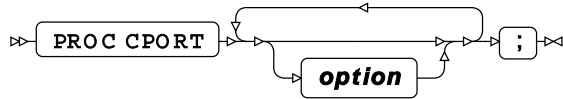
CPORT procedure

Supported statements

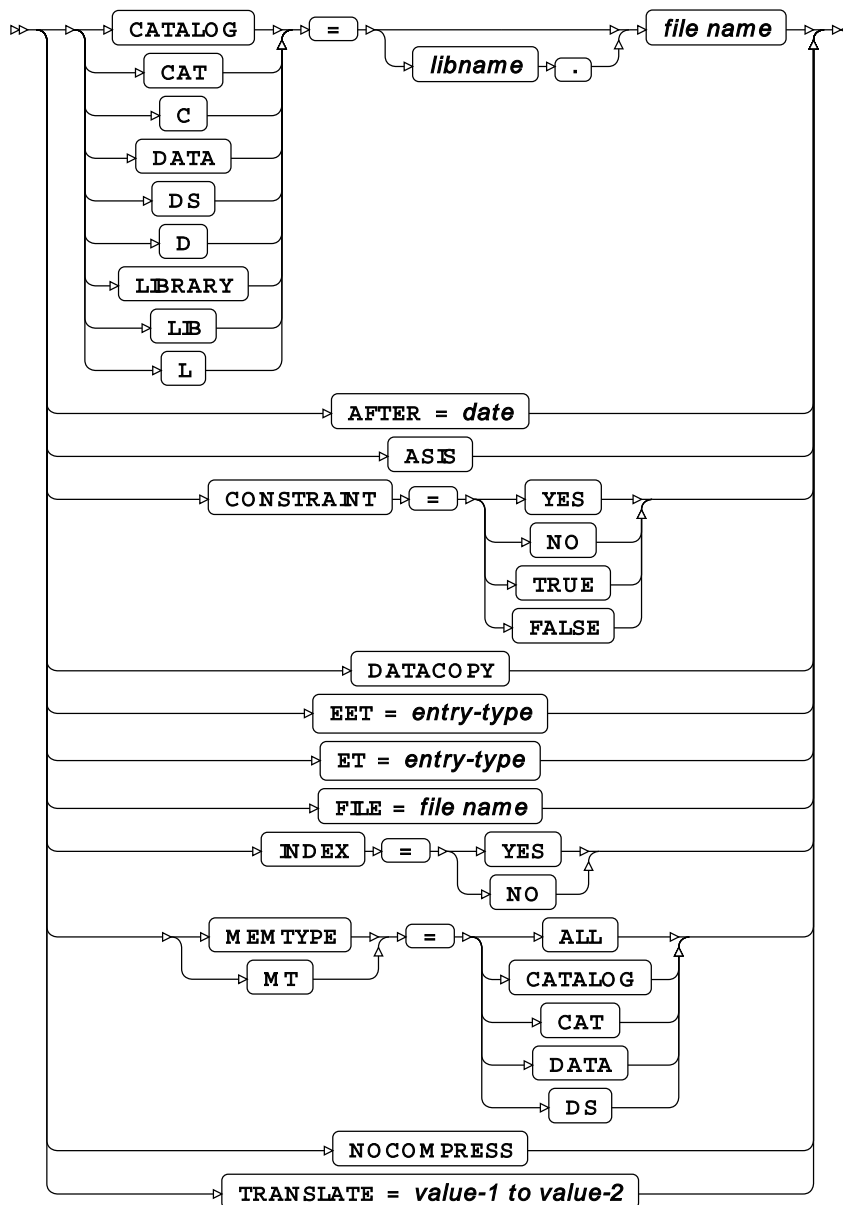
- *PROC CPORT* [↗](#) (page 2311)
- *EXCLUDE* [↗](#) (page 2312)
- *SELECT* [↗](#) (page 2312)

PROC CPORT

Creates a transport file containing a backup of datasets.

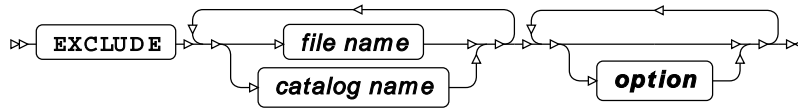


option

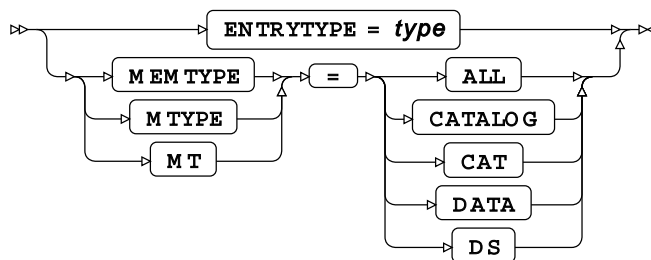


EXCLUDE

Excludes the specified members.

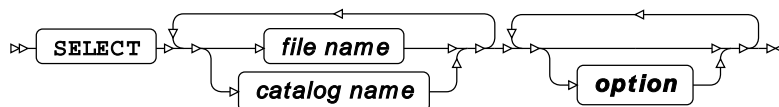


option

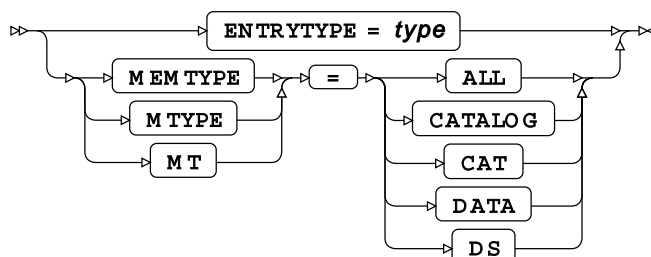


SELECT

Includes only the specified members.



option



DATASETS procedure

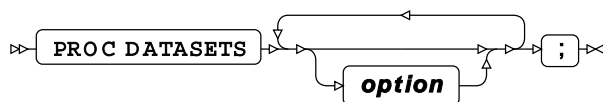
Supported statements

- `PROC DATASETS` [↗](#) (page 2313)

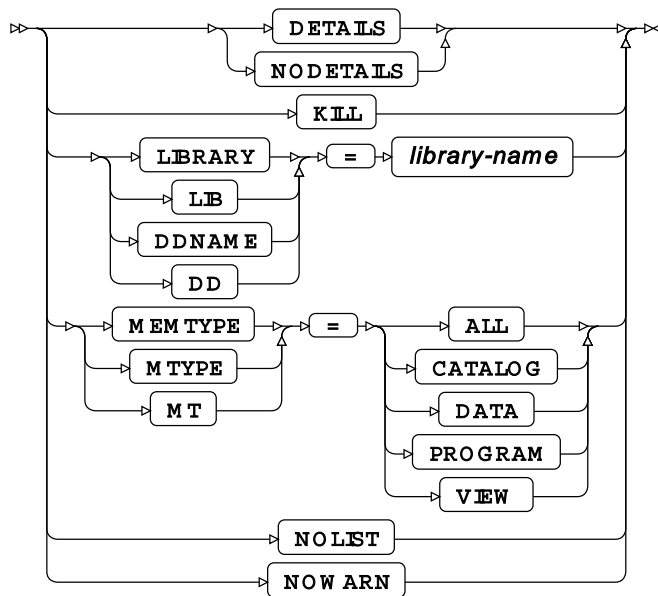
- *AGE* [↗](#) (page 2314)
- *APPEND* [↗](#) (page 2315)
- *CHANGE* [↗](#) (page 2315)
- *CONTENTS* [↗](#) (page 2316)
- *COPY* [↗](#) (page 2316)
- *DELETE* [↗](#) (page 2317)
- *EXCLUDE* [↗](#) (page 2317)
- *EXCHANGE* [↗](#) (page 2318)
- *FORMAT* [↗](#) (page 2318)
- *INDEX CREATE* [↗](#) (page 2319)
- *INDEX DELETE* [↗](#) (page 2319)
- *INFORMAT* [↗](#) (page 2319)
- *LABEL* [↗](#) (page 2319)
- *MODIFY* [↗](#) (page 2320)
- *RENAME* [↗](#) (page 2320)
- *REPAIR* [↗](#) (page 2320)
- *SELECT* [↗](#) (page 2320)

PROC DATASETS

Manages libraries.

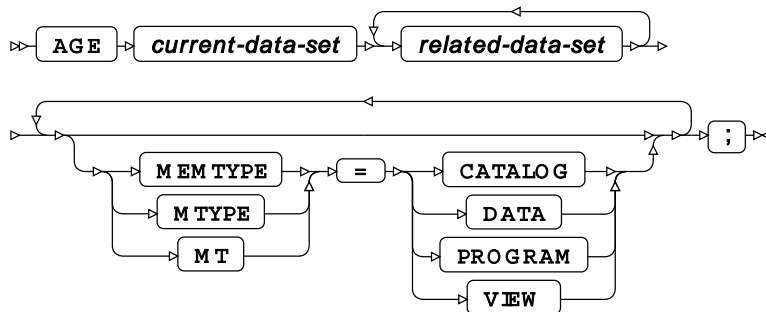


option



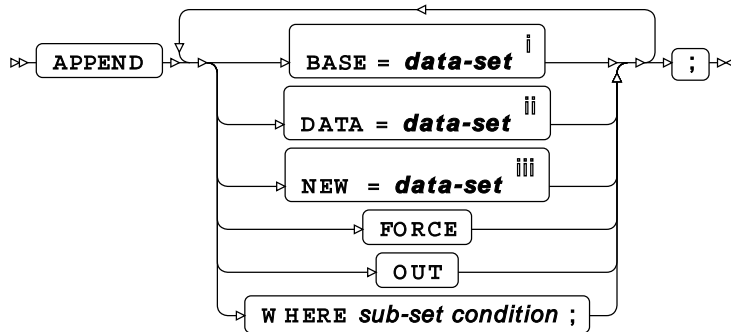
AGE

Maintains historical data.



APPEND

Appends observations from a dataset to a different existing dataset.



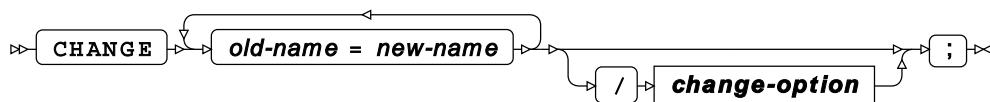
ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

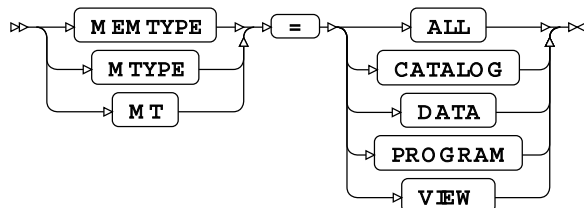
ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

CHANGE

Changes the name of a member in a library.

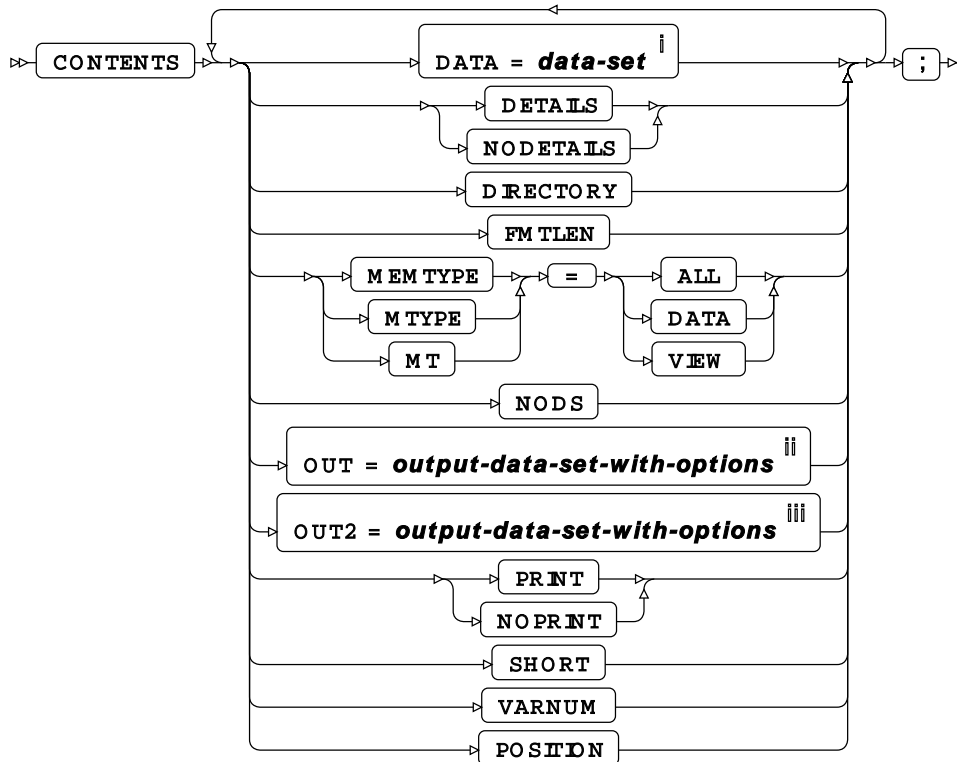


change-option



CONTENTS

Describes one or more datasets in a library.



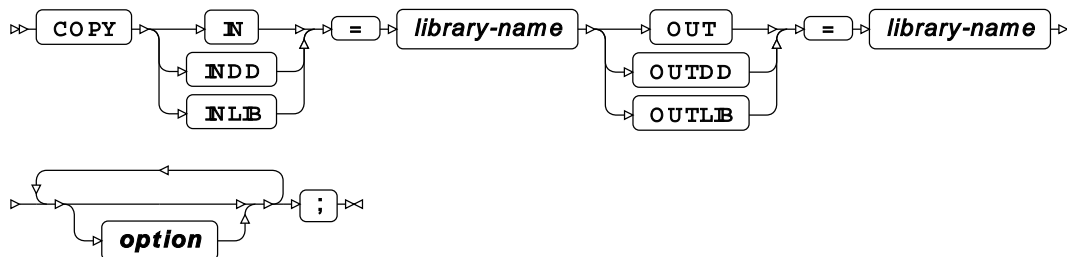
ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

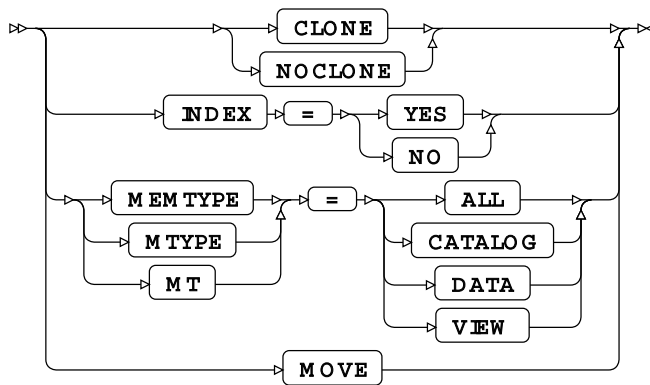
ⁱⁱⁱ See *Output dataset* [↗](#) (page 16).

COPY

Copies one or more members in a library to a different library.

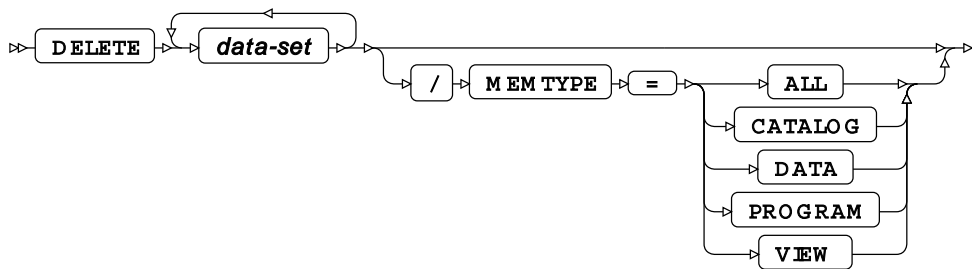


option



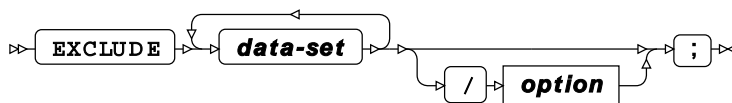
DELETE

Deletes one or more members.

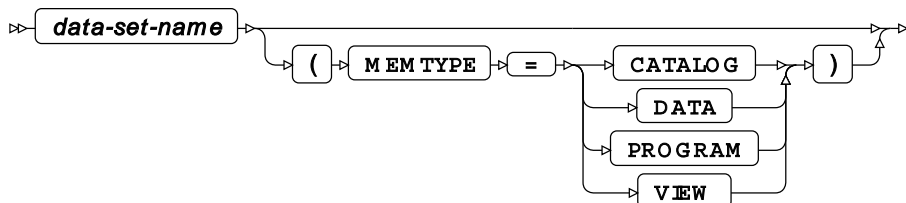


EXCLUDE

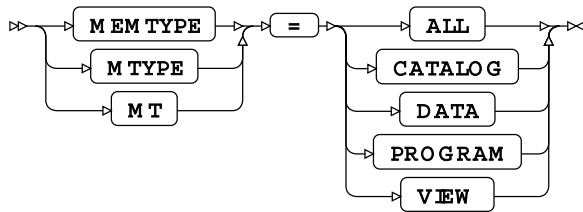
Includes all members except those listed.



data-set

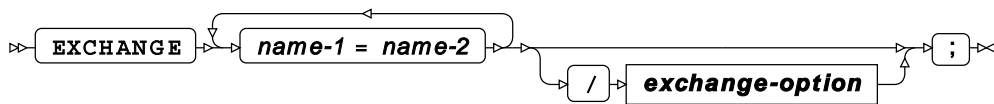


option

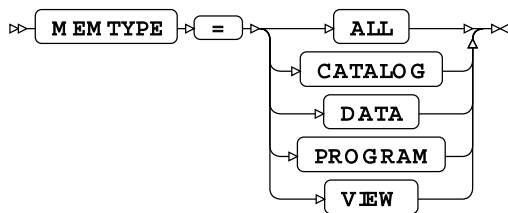


EXCHANGE

Exchanges items within a dataset.



exchange-option



FORMAT

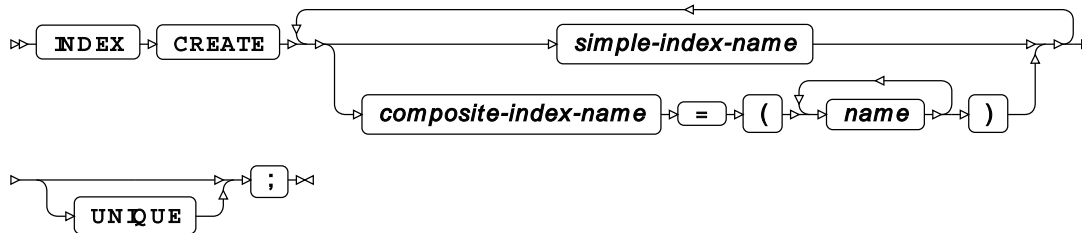
Adds formats to one or more variables in a dataset.



ⁱ See *Variable Lists* [↗](#) (page 32).

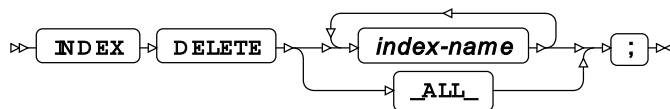
INDEX CREATE

Creates an index based on a simple-index-name, or composite-index-name.



INDEX DELETE

Deletes one or more members.



INFORMAT

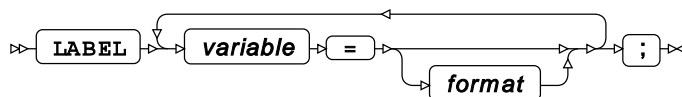
Adds informats to one or more variables in a dataset.



ⁱ See *Variable Lists* [↗](#) (page 32).

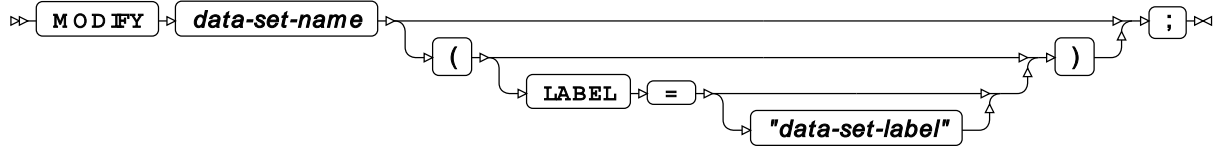
LABEL

Adds labels to one or more variables in a dataset.



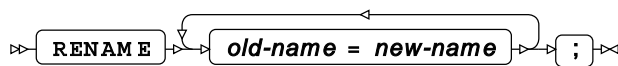
MODIFY

Modifies the metadata of the specified dataset.



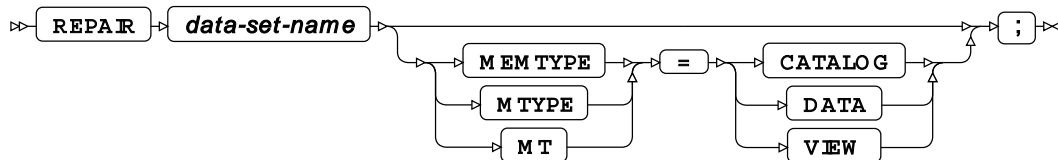
RENAME

Renames one or more variables in a dataset.



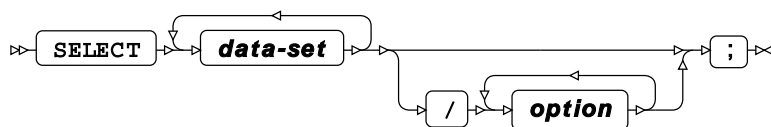
REPAIR

Recreates the index files of a dataset, if they have been deleted.

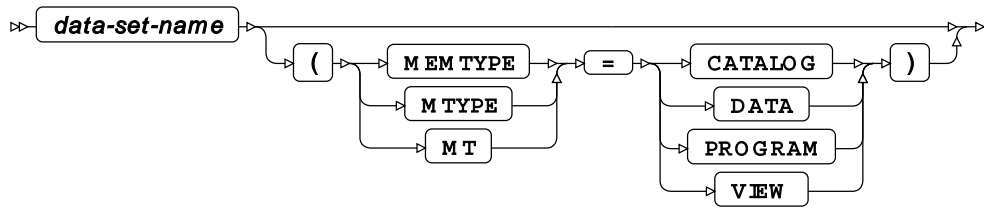


SELECT

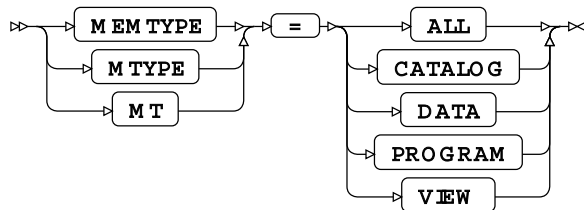
Includes only those specified members.



data-set



option



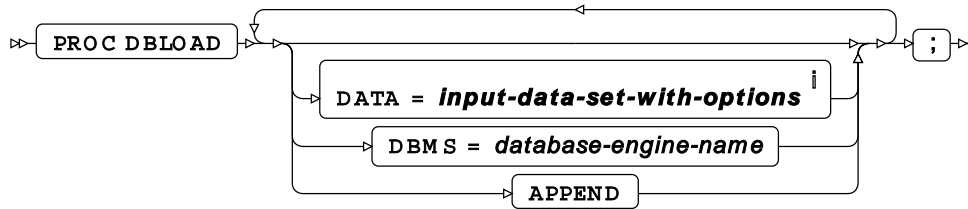
DBLOAD procedure

Supported statements

- *PROC DBLOAD* [↗](#) (page 2322)
- *ACCDESC* [↗](#) (page 2322)
- *COLUMN* [↗](#) (page 2322)
- *COMMIT* [↗](#) (page 2322)
- *DELETE* [↗](#) (page 2323)
- *ERRLIMIT* [↗](#) (page 2323)
- *LABEL* [↗](#) (page 2323)
- *LIMIT* [↗](#) (page 2323)
- *LIST* [↗](#) (page 2323)
- *LOAD* [↗](#) (page 2324)
- *NULL* [↗](#) (page 2324)
- *RENAME* [↗](#) (page 2324)
- *RESET* [↗](#) (page 2324)
- *SQL* [↗](#) (page 2324)
- *TABLE* [↗](#) (page 2325)
- *TYPE* [↗](#) (page 2325)
- *WHERE* [↗](#) (page 2325)

PROC DBLOAD

Loads data into a database.



ⁱ See *Input dataset* [↗](#) (page 16).

ACCDESC

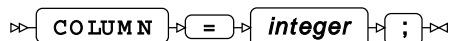
Creates an access descriptor based on the connection information.



ⁱ See *Access Descriptors* [↗](#) (page 15).

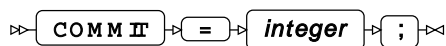
COLUMN

Renames one or more variables in a dataset. This statement is an alias for **RENAME**.



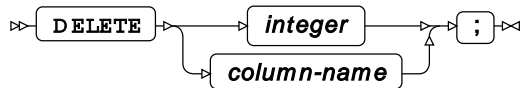
COMMIT

Specifies how many operations you have to perform before a **COMMIT** statement.



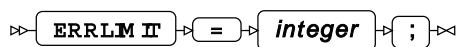
DELETE

Drops a column from the input dataset.



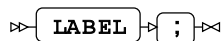
ERRLIMIT

Specifies the number of errors to be generated before stopping the task.



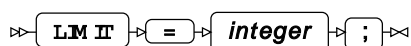
LABEL

Uses column labels rather than names when generating database column names.



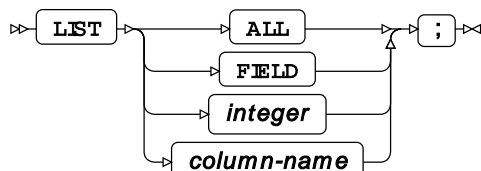
LIMIT

Maximises the number of rows to insert.



LIST

Lists out information about the variables of the data being inserted.



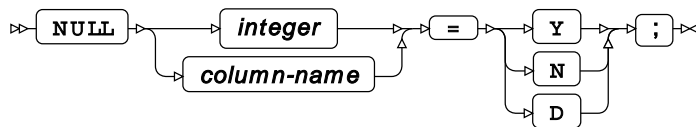
LOAD

Starts the insertion process.



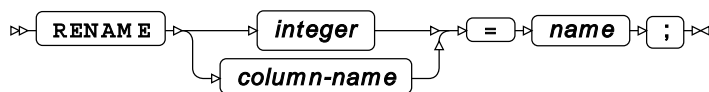
NULL

Specifies which column can be **NULL** when the database table is being created.



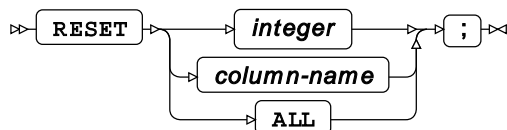
RENAME

Renames input columns before insertion of data.



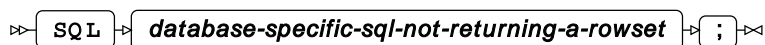
RESET

Resets column details back to their default values.



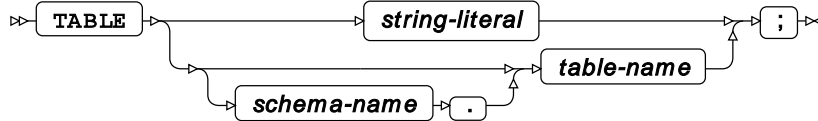
SQL

Submits unmodified database-specific SQL statements to the database.



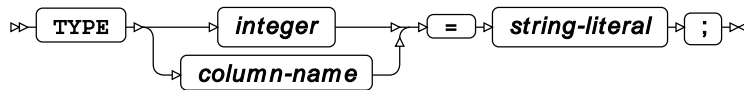
TABLE

Specifies the name of the table that data will be inserted into.



TYPE

Sets database specific type for one or more columns.



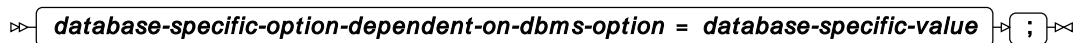
WHERE

Restricts the observations to be processed.



Connection option

Lists connection options for the specified DBMS. These options include, username, password, and so on.



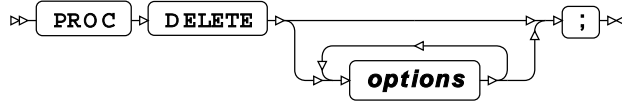
DELETE procedure

Supported statements

- *PROC DELETE* [↗](#) (page 2326)

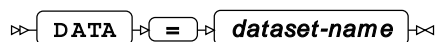
PROC DELETE

Deletes one or more datasets from a library.



Options

DATA



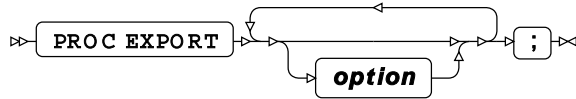
EXPORT procedure

Supported statements

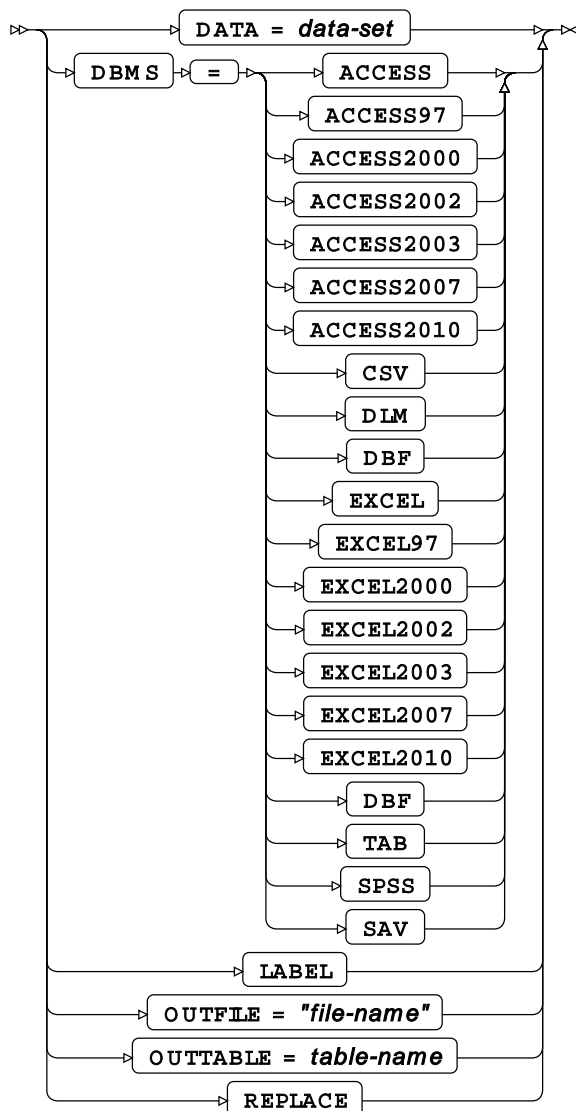
- *PROC EXPORT* [↗](#) (page 2327)
- *DATABASE* [↗](#) (page 2328)
- *DBLIBOPTS* [↗](#) (page 2328)
- *DBPASSWORD* [↗](#) (page 2328)
- *DELIMITER* [↗](#) (page 2328)
- *MENGINE* [↗](#) (page 2328)
- *NEWFILE* [↗](#) (page 2329)
- *PASSWORD* [↗](#) (page 2329)
- *PUTNAMES* [↗](#) (page 2329)
- *SHEET* [↗](#) (page 2329)
- *USER* [↗](#) (page 2329)
- *WGDB* [↗](#) (page 2330)

PROC EXPORT

Exports a WPS dataset to the specified DBMS file.

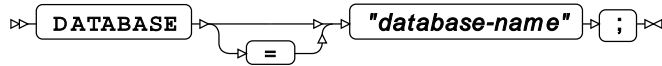


option



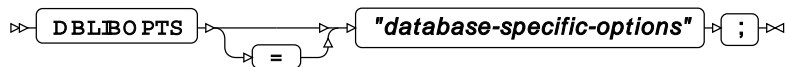
DATABASE

Specifies the database only when DBMS = MS Access.



DBLIBOPTS

Specifies database-specific connection options.



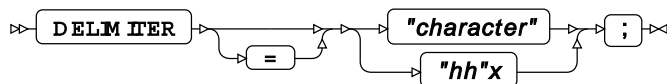
DBPASSWORD

Specifies the password for a protected Microsoft Access file.



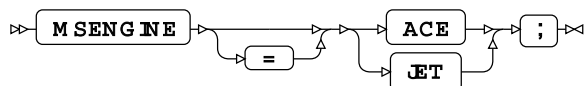
DELIMITER

Specifies a delimiter character for delimited text files.



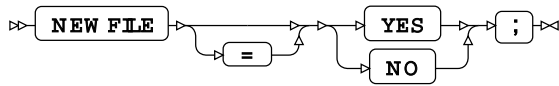
MENGINE

Specifies the Microsoft engines to use if Microsoft Excel is specified as the export database.



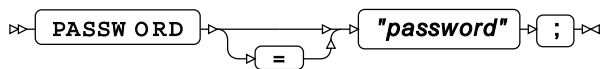
NEWFILE

Specifies whether the file is new or not new.



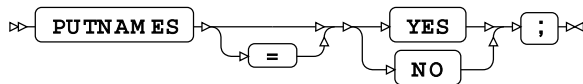
PASSWORD

Specifies the password for a protected Microsoft Access file.



PUTNAMES

Outputs from a text file or MS Excel statement.



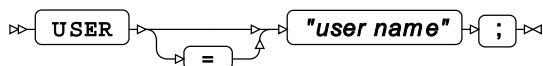
SHEET

Specifies the name of the worksheet in the specified Microsoft Excel workbook into which data is exported.



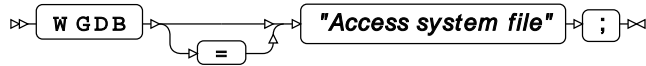
USER

Specifies the user name for a protected Microsoft Access database.



WGDB

Specifies the name of a workgroup database access system file.



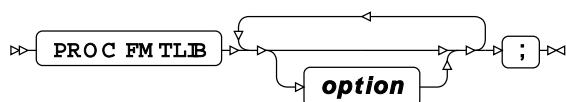
FMTLIB procedure

Supported statements

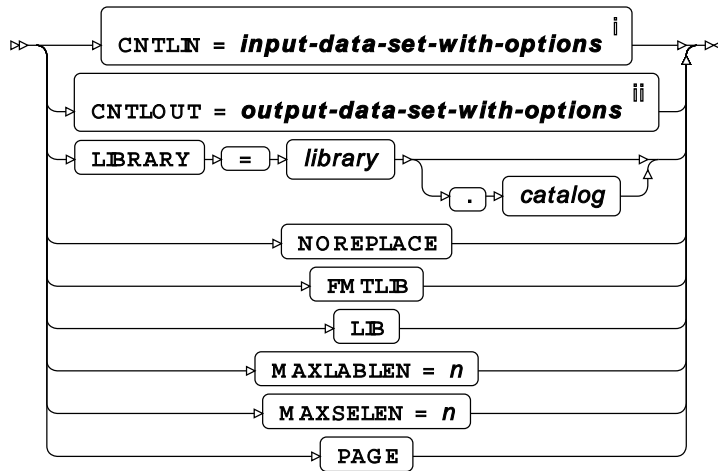
- *PROC FMTLIB* [↗](#) (page 2330)
- *EXCLUDE* [↗](#) (page 2331)
- *INVALUE* [↗](#) (page 2331)
- *PICTURE* [↗](#) (page 2332)
- *SELECT* [↗](#) (page 2333)
- *VALUE* [↗](#) (page 2333)

PROC FMTLIB

Manages user-defined format catalogs.



option

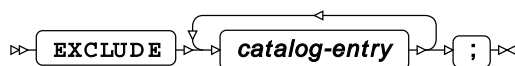


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

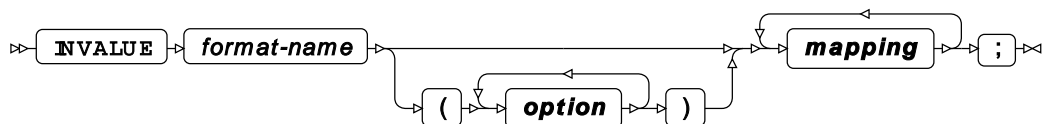
EXCLUDE

Excludes entries being processed by the procedure.

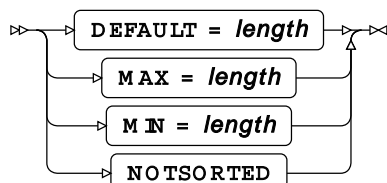


INVALUE

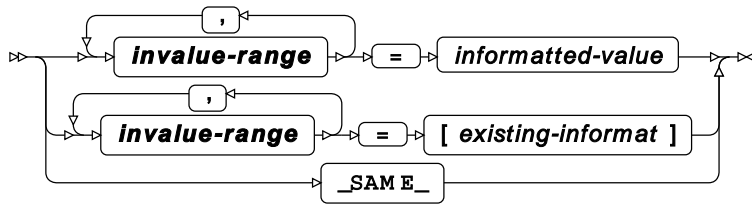
Creates a user-defined informat.



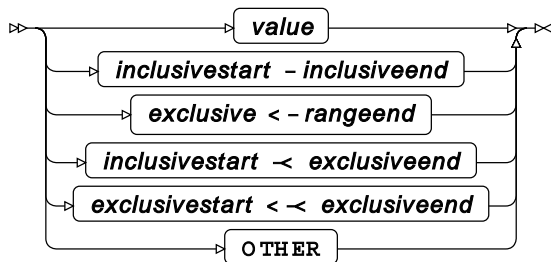
option



mapping

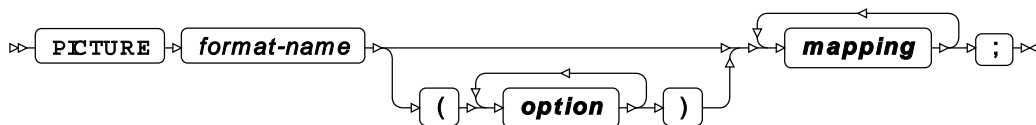


invalue-range

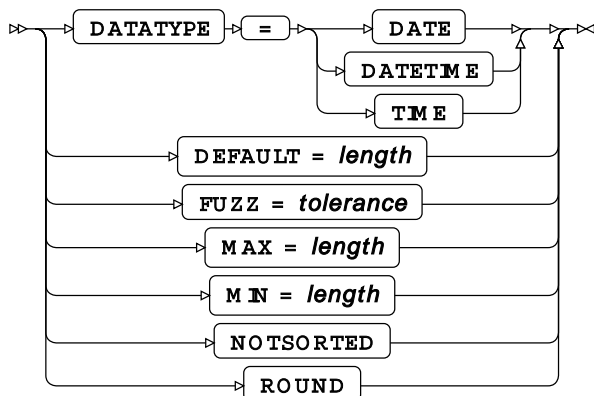


PICTURE

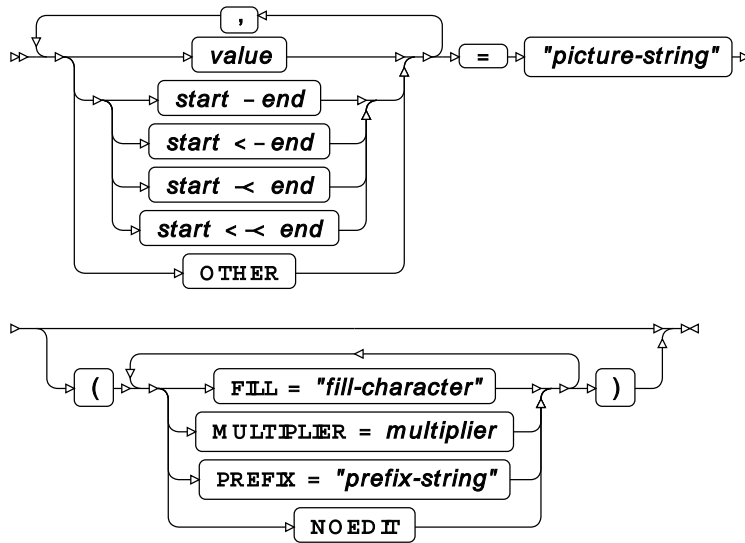
Creates a user-defined picture format.



option

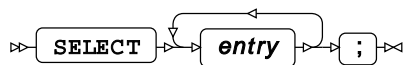


mapping



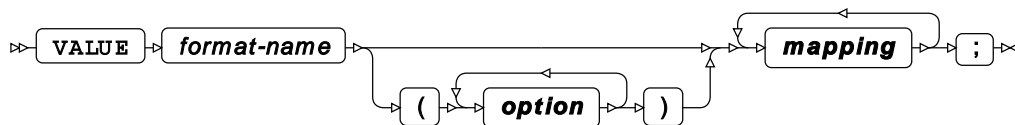
SELECT

Selects entries to be processed by the procedure.

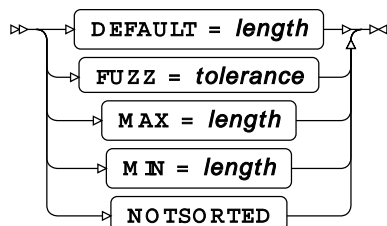


VALUE

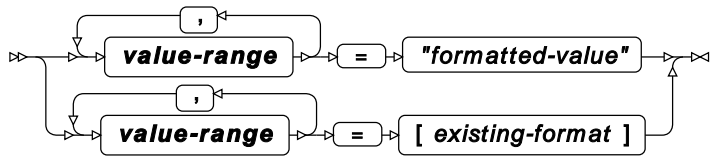
Creates a user-defined format.



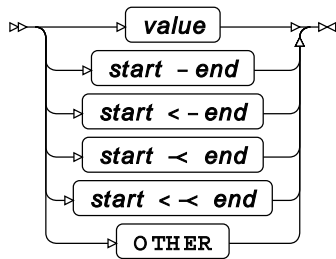
option



mapping



value-range



FONT Procedure

Returns information about TrueType fonts that can be used in WPS Analytics.

The `FONT` procedure reports the details of how WPS Analytics is configured to use TrueType fonts. The report contains the following tables:

- Font Configuration Files. Lists the top level TrueType font configuration files used by WPS Analytics.
- Font Directories. Lists all folders that are searched for font files. On z/OS systems, this table also includes MVS datasets.
- Available Fonts. Lists all font files found in the searched font folders. On z/OS systems, this table also includes MVS datasets.

The available TrueType fonts are used by the Output Delivery System (ODS) `GRAPHICS` and `PDF` destinations in SAS language programs run in WPS Analytics.

Examples ↗	2335
Examples of using the <code>FONT</code> procedure.	
<code>FONT</code> procedure reference ↗	2337
Describes the syntax and options for <code>PROC FONT</code> .	

Examples

Examples of using the FONT procedure.

Basic example [↗](#)..... 2335

The following example returns the default level of information about available TrueType fonts.

Example – more detailed font configuration information [↗](#)..... 2336

The following example returns more detailed information about the available TrueType fonts.

Basic example

The following example returns the default level of information about available TrueType fonts.

```
PROC FONT;
```

This produces the following tables that are written to ODS output.

Font Configuration Files

This table lists the top level TrueType font configuration file used by WPS Analytics

Obs	Configuration Filename
1	C:\Program Files\WPS\etc\fonts\fonts.conf

Font Directories

This table lists all folders that are searched for font files.

Obs	Directory Name
1	C:/Windows/fonts
2	C:/Users/.../.local/share/fonts
3	C:/Users/.../.fonts

Available Fonts

This table lists the TrueType font files and associated font family found in the searched font folders. A sub-set of the output is shown below.

Obs	Filename	Font Family
1	C:/Windows/fonts/ANTQUAB.TTF	Book Antiqua
2	C:/Windows/fonts/ANTQUABI.TTF	Book Antiqua
3	C:/Windows/fonts/arial.ttf	Arial
4	C:/Windows/fonts/arialbd.ttf	Arial
5	C:/Windows/fonts/ARIALN.TTF	Arial
6	C:/Windows/fonts/ARIALN.TTF	Arial Narrow
7	C:/Windows/fonts/ARIALNBI.TTF	Arial

```

8      C:/Windows/fonts/ARIALNBI.TTF      Arial Narrow
9      C:/Windows/fonts/arialbk.ttf       Arial
10     C:/Windows/fonts/arialbk.ttf       Arial Black
...
```

Example – more detailed font configuration information

The following example returns more detailed information about the available TrueType fonts.

```
PROC FONT NOSHORT;
```

This produces the same tables written to ODS output as the basic example, with additional information in the **Font Configuration Files** and **Available Fonts** tables.

Font Configuration Files

This table lists all TrueType font configuration files used by WPS Analytics. A sub-set of the output is shown below.

Obs	Configuration Filename
1	C:\Program Files\WPS\etc\fonts\fonts.conf
2	C:\Program Files\WPS\etc\fonts\conf.d
3	C:\Program Files\WPS\etc\fonts\conf.d/10-scale-bitmap-fonts.conf
4	C:\Program Files\WPS\etc\fonts\conf.d/20-unhint-small-vera.conf
...	
15	C:\Program Files\WPS\etc\fonts\conf.d/69-unifont.conf
16	C:\Program Files\WPS\etc\fonts\conf.d/80-delicious.conf
17	C:\Program Files\WPS\etc\fonts\conf.d/90-synthetic.conf

Available Fonts

This table lists the TrueType font files, associated font family, and the full name of the font found in the searched font folders. A sub-set of the output is shown below.

Obs	Filename	Font Family	Full Name
...			
298	C:/Windows/fonts/Vera.ttf	Bitstream Vera Sans	Bitstream Vera Sans
299	C:/Windows/fonts/VeraBd.ttf	Bitstream Vera Sans	Bitstream Vera Sans Bold
300	C:/Windows/fonts/VeraBI.ttf	Bitstream Vera Sans	Bitstream Vera Sans Bold Oblique
301	C:/Windows/fonts/VeraIt.ttf	Bitstream Vera Sans	Bitstream Vera Sans Oblique
302	C:/Windows/fonts/VeraMoBd.ttf	Bitstream Vera Sans Mono	Bitstream Vera Sans Mono Bold
303	C:/Windows/fonts/VeraMoBI.ttf	Bitstream Vera Sans Mono	Bitstream Vera Sans Mono Bold Oblique
304	C:/Windows/fonts/VeraMoIt.ttf	Bitstream Vera Sans Mono	Bitstream Vera Sans Mono Oblique
305	C:/Windows/fonts/VeraMono.ttf	Bitstream Vera Sans Mono	Bitstream Vera Sans Mono

306	C:/Windows/fonts/VeraSe.ttf	Bitstream Vera Serif	Bitstream Vera Serif
307	C:/Windows/fonts/VeraSeBd.ttf	Bitstream Vera Serif	Bitstream Vera Serif Bold
...			

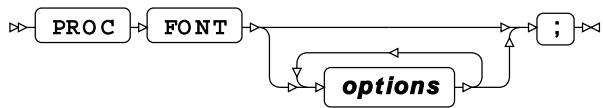
FONT procedure reference

Describes the syntax and options for PROC FONT.

PROC FONT ↗	2337
Enables you to return information for installed TrueType fonts and specify the level of detail returned.	

PROC FONT

Enables you to return information for installed TrueType fonts and specify the level of detail returned.



You can specify that the font information returned is detailed or concise, and which tables in the FONT procedure output are displayed.

Specifying PROC FONT; with no options is the same as specifying

```
PROC FONT CONFIG FONTDIRS FONTS SHORT;
```

Options

The following options are available with the PROC FONT statement

CONFIG

Returns information about the font configuration files used by WPS Analytics.



Type: Keyword

The list of configuration files is displayed in the **Font Configuration Files** table in the FONT procedure output. The default output includes this table.

- If SHORT is specified, only the location of the top-level font configuration file is listed.
- If NOSHORT is specified, all font configuration files are listed.

The **Font Configuration Files** table has the following format:

Obs	Configuration Filename
count	file-path

Where:

- *count* is an incrementing count of the folders searched.
- *file-path* is the absolute path the font configuration file.

FONTDIRS

Returns a list of the folder paths searched for font files.

➤ **FONTDIRS** ➤

Type: Keyword

The list of folder paths is displayed in the **Font Directories** table in the `FONT` procedure output. On z/OS, the table displays the path of all files and MVS datasets searched. The default output includes this table.

The **Font Directories** table has the following format:

Obs	Directory Name
count	folder-path

Where:

- *count* is an incrementing count of the folders searched.
- *folder-path* is the absolute path of the searched folder.

FONTS

Returns a list of the fonts available for use by WPS Analytics.

➤ **FONTS** ➤

Type: Keyword

The list of fonts is displayed in the **Available Fonts** table in the `FONT` procedure output. On z/OS, the table displays the path of all files and MVS datasets searched. The default output includes this table.

- If `SHORT` is specified, the path of the font file and the family (typeface) to which the font belongs are displayed.
- If `NOSHORT` is specified, the full name of the font family is also displayed.

The **Available Fonts** table has the following format:

Obs	Filename	Font Family	Full Name
count	file-path	family-name	font-name

Where:

- *count* is an incrementing count of the font files available in the searched font folders.
- *file-path* is the absolute path of the font definition file.
- *family-name* is the generic name for the font. The same generic font name might apply to multiple entries in the table.
- *font-name* is the specific name for the font in the font family.

NOCONFIG

No information about configuration files is returned.

➤ NOCONFIG ➤

Type: Keyword

The **Font Configuration Files** table is not displayed in the FONT procedure output.

NOFONDIRS

No information about font directories is returned.

➤ NOFONDIRS ➤

Type: Keyword

The **Font Directories** table is not displayed in the FONT procedure output.

NOFONTS

No information about available fonts is returned.

➤ NOFONTS ➤

Type: Keyword

The **Available Fonts** table is not displayed in the FONT procedure output.

NOSHORT

Specifies an increased level of information is displayed in the FONT procedure output.

➤ NOSHORT ➤

Type: Keyword

When specified:

- The **Font Configuration Files** table displays all font configuration files.
- The **Available Fonts** table contains the full name column in addition to the filename and font family columns.

SHORT

Specifies that a reduced level of information is displayed in the `FONT` procedure output. This is the default level of detail returned.



Type: Keyword

When specified:

- The **Font Configuration Files** table only displays the op-level font configuration file.
- The **Available Fonts** table does not contain the full name column.

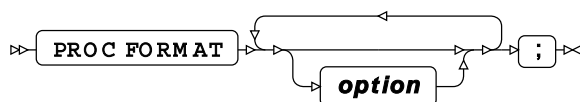
FORMAT procedure

Supported statements

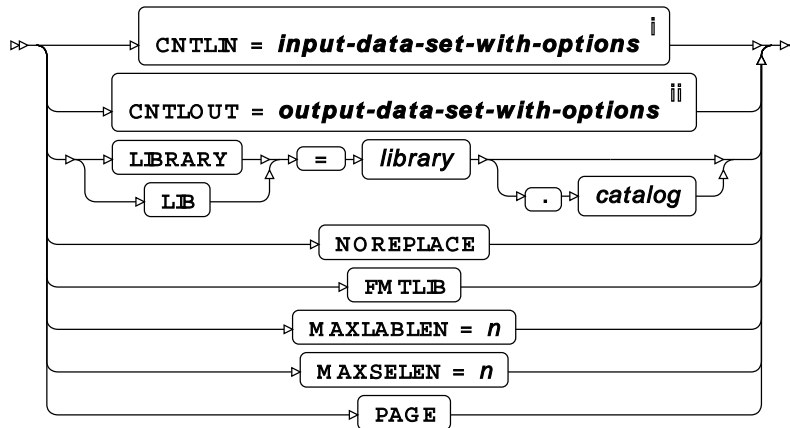
- `PROC FORMAT` [↗](#) (page 2340)
- `EXCLUDE` [↗](#) (page 2331)
- `INVALUE` [↗](#) (page 2331)
- `PICTURE` [↗](#) (page 2332)
- `SELECT` [↗](#) (page 2333)
- `VALUE` [↗](#) (page 2333)

PROC FORMAT

Manages user-defined format catalogs.



option

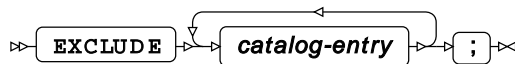


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

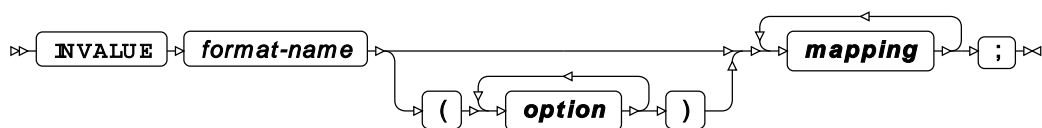
EXCLUDE

Excludes entries being processed by the procedure.

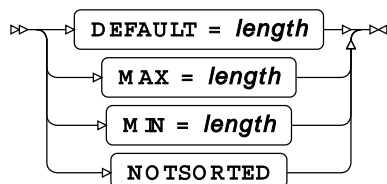


INVALUE

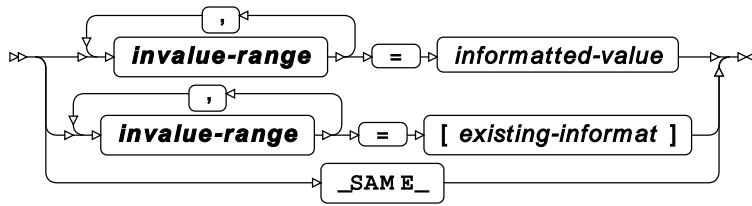
Creates a user-defined informat.



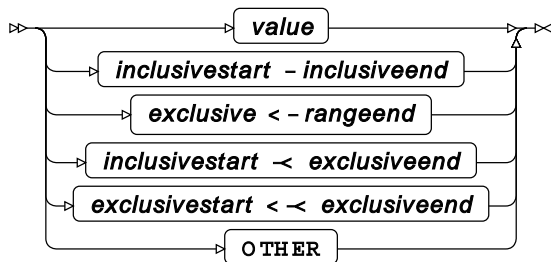
option



mapping

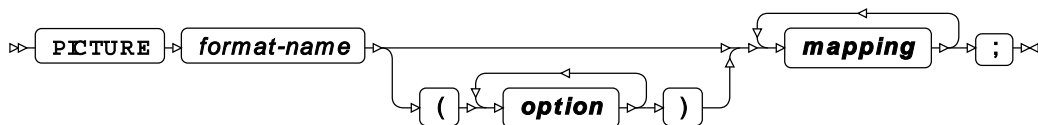


invalue-range

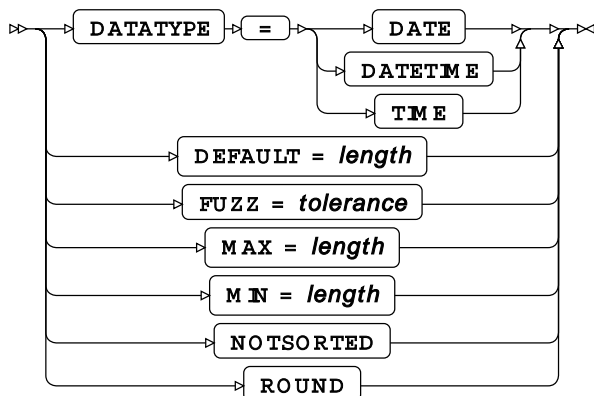


PICTURE

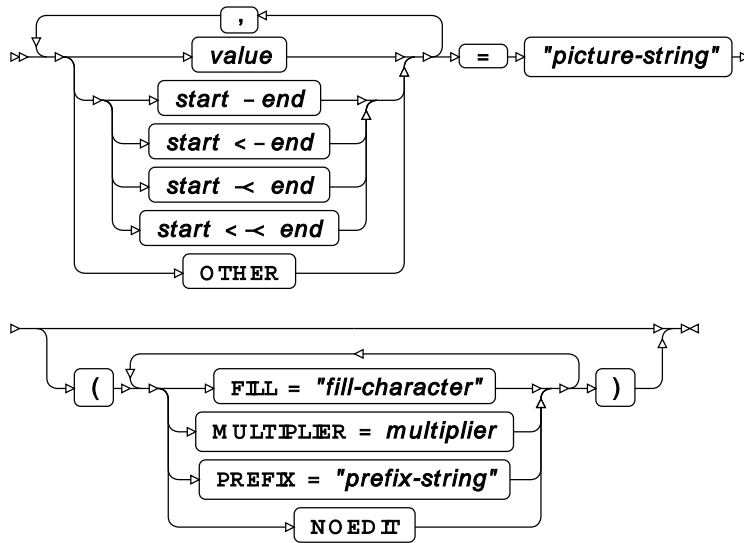
Creates a user-defined picture format.



option

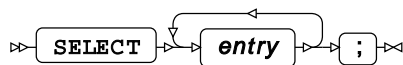


mapping



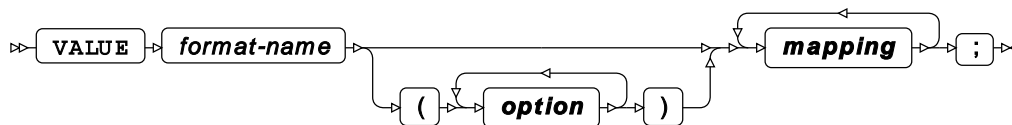
SELECT

Selects entries to be processed by the procedure.

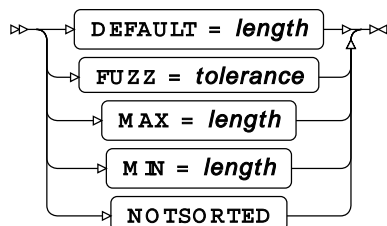


VALUE

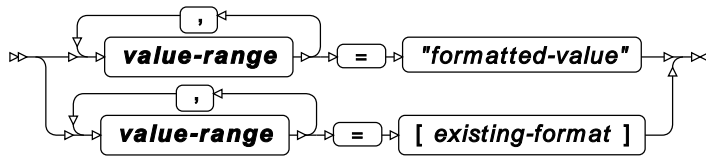
Creates a user-defined format.



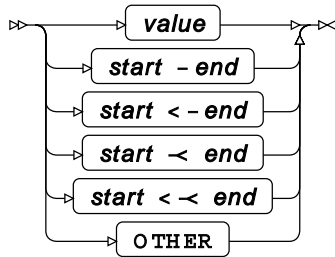
option



mapping



value-range



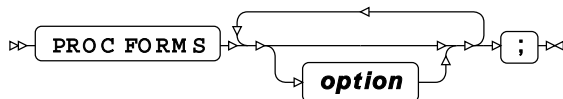
FORMS procedure

Supported statements

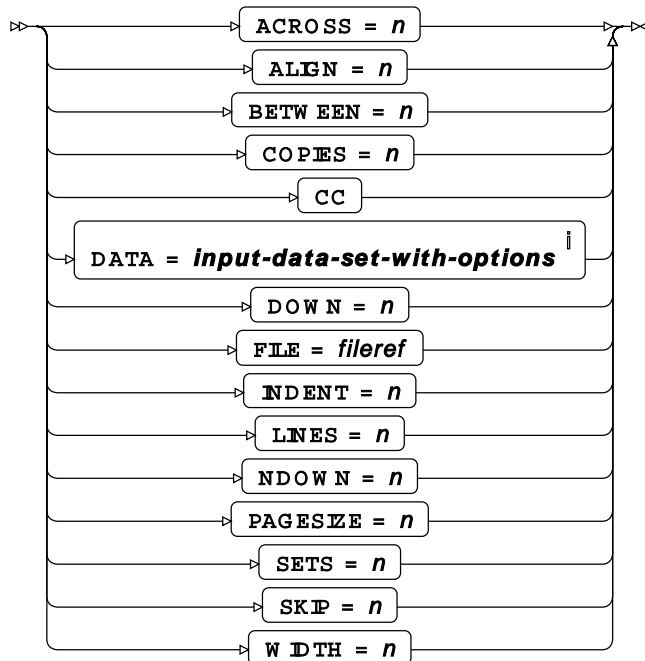
- *PROC FORMS* [↗](#) (page 2344)
- *BY* [↗](#) (page 2345)
- *FREQ* [↗](#) (page 2345)
- *LINE* [↗](#) (page 2346)
- *WHERE* [↗](#) (page 2346)

PROC FORMS

Processes mail merging layout data that populates forms.



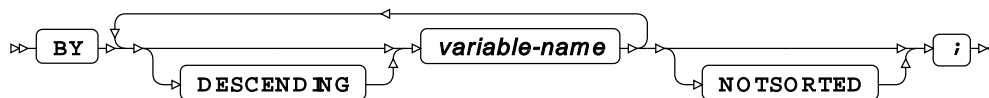
option



ⁱ See *Input dataset* [↗](#) (page 16).

BY

Groups the observations in a dataset using one or more specified variables.

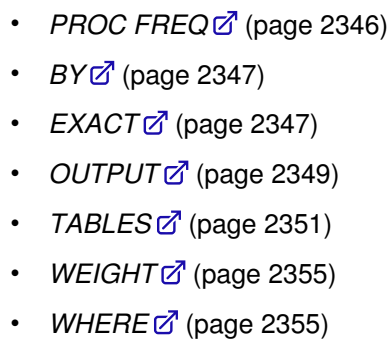


FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



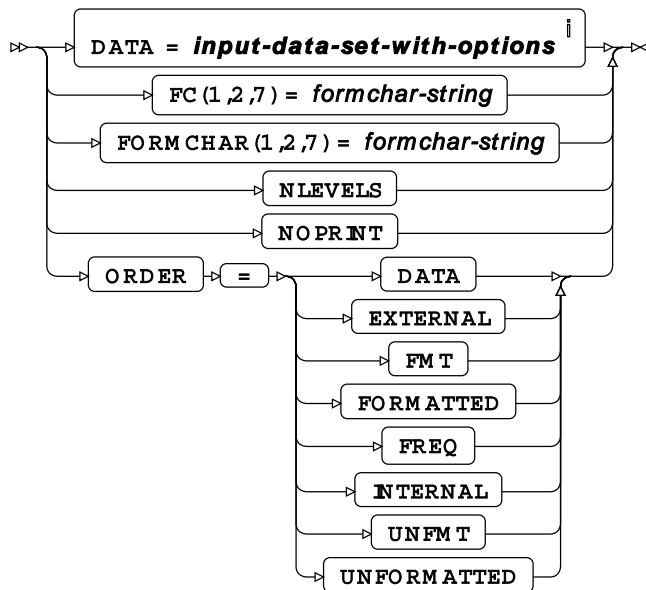
Structures in lines, which provides the format for the fields of data.



Calculates frequency tables from a given dataset.



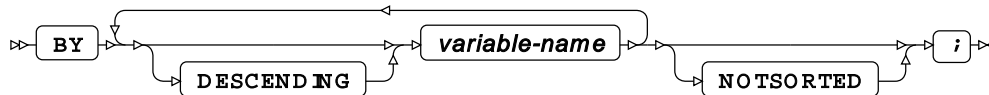
option



ⁱ See [Input dataset](#) (page 16).

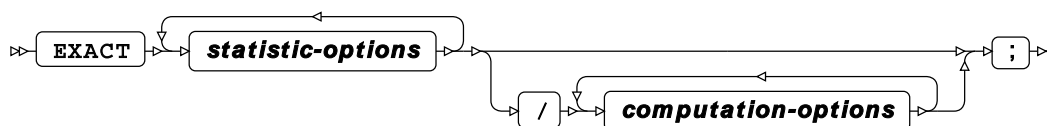
BY

Groups the observations in a dataset using one or more specified variables.

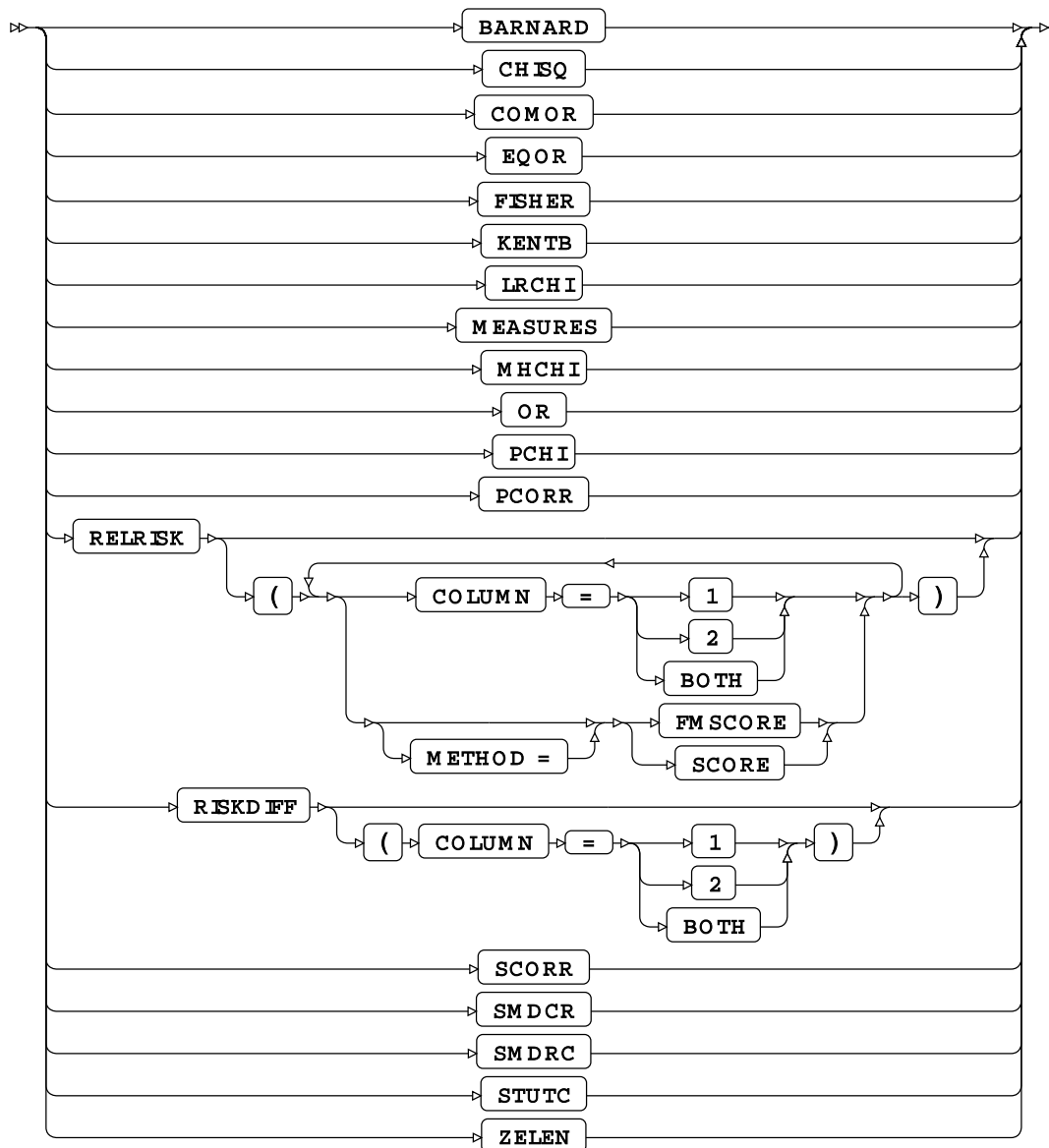


EXACT

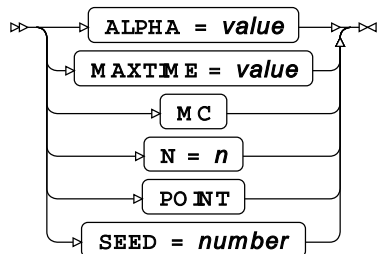
Specifies the type of statistic to be calculated for tables.



statistic-options

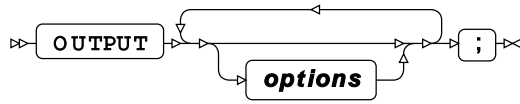


computation-options

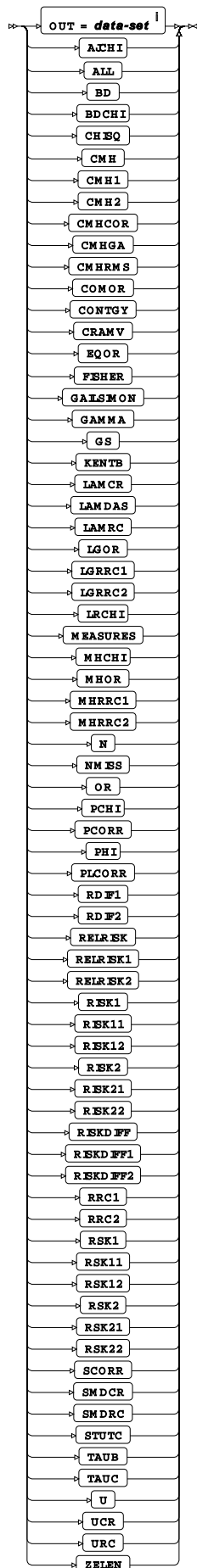


OUTPUT

Outputs the last table specified to a dataset.



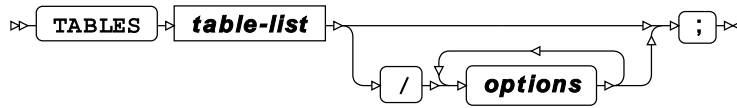
options



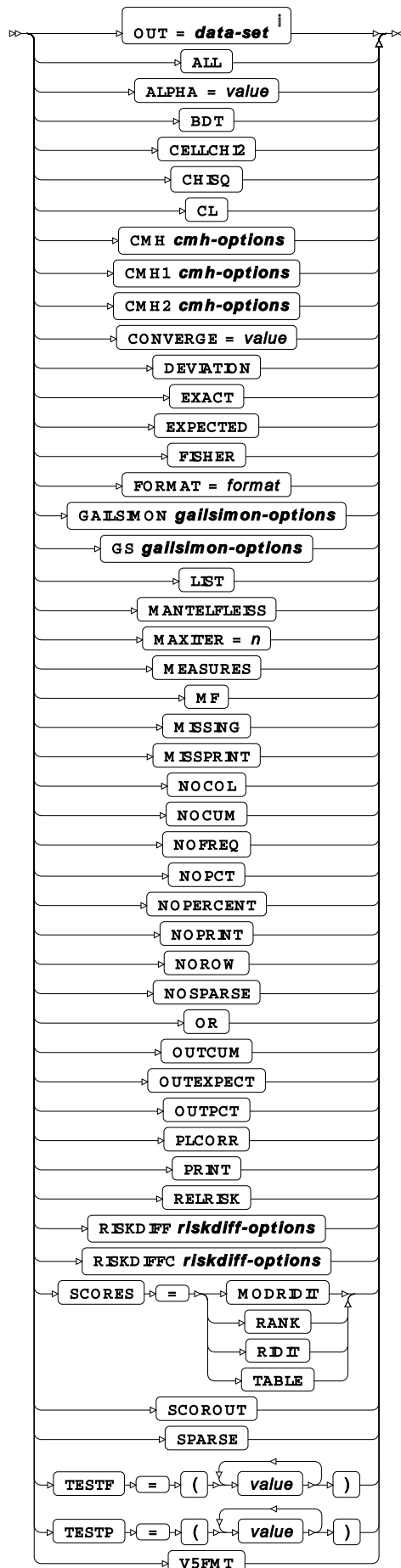
ⁱ See *Output dataset* [↗](#) (page 16).

TABLES

Lists frequency tables and statistics to be computed for each.

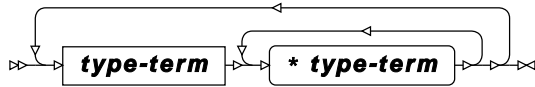


options

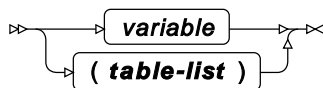


ⁱ See *Output dataset* [↗](#) (page 16).

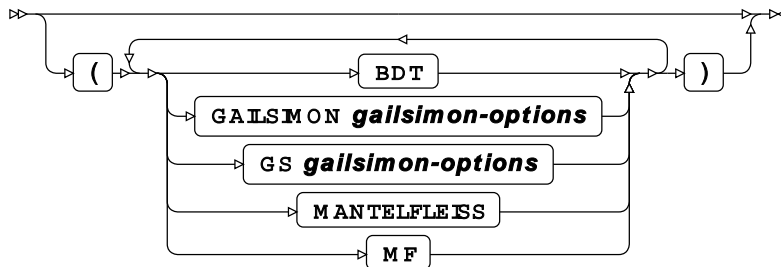
table-list



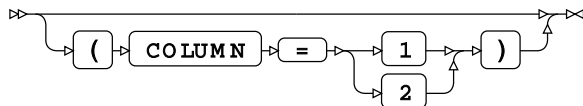
type-term



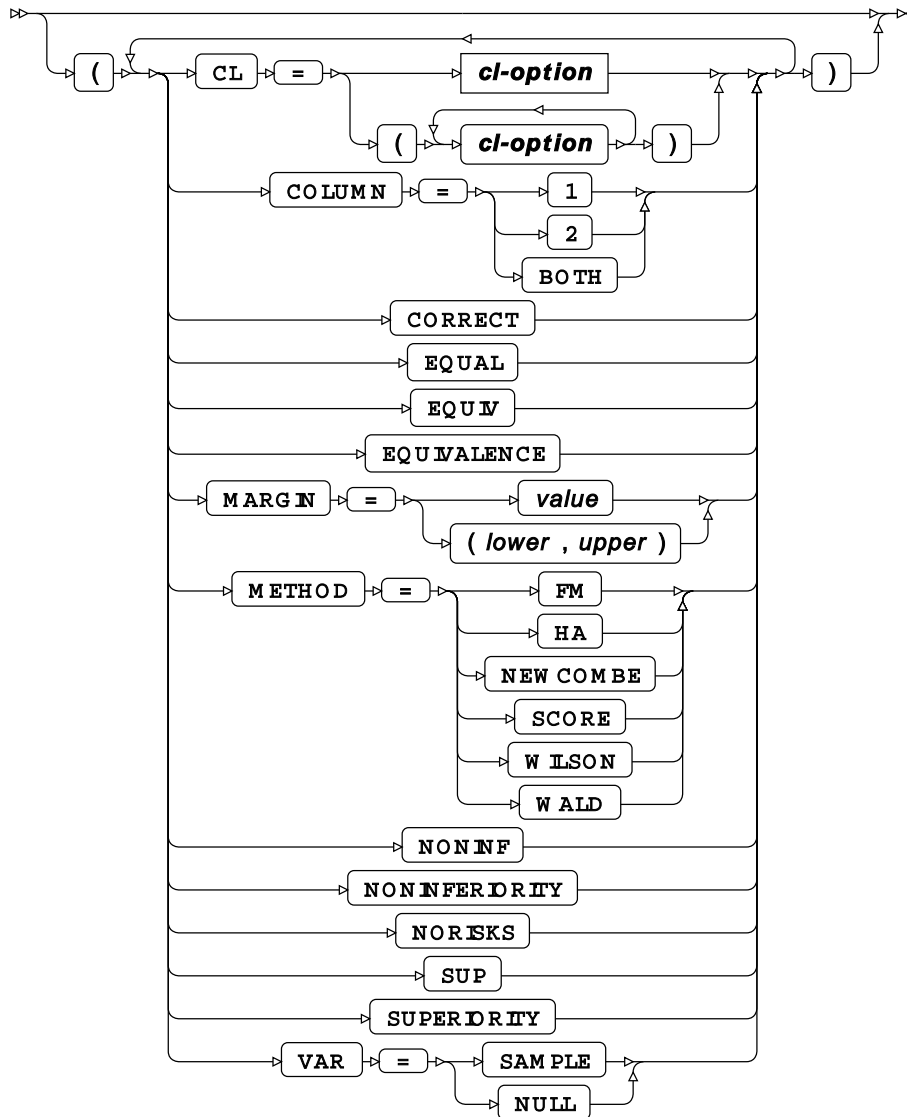
cmh-options



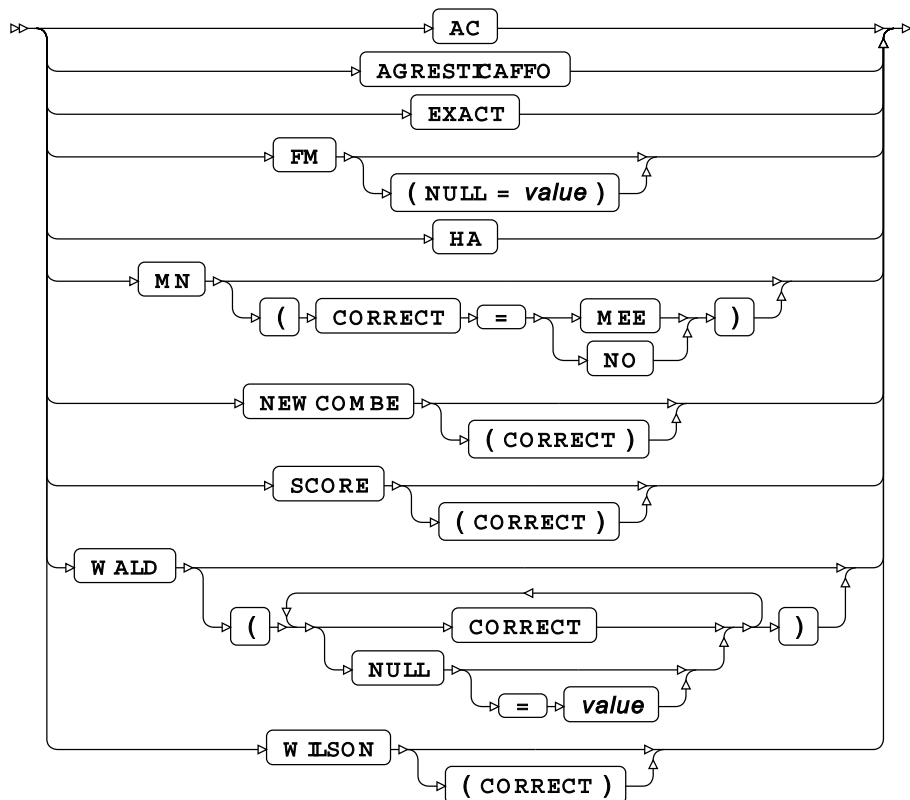
gailsimon-options



riskdiff-options

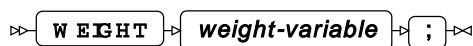


cl-option



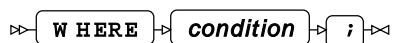
WEIGHT

Specifies a variable giving the weight associated with each observation.



WHERE

Restricts the observations to be processed.



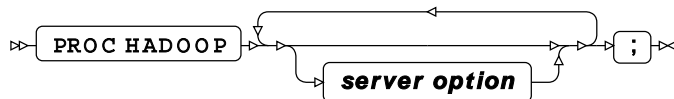
HADOOP Procedure

Supported statements

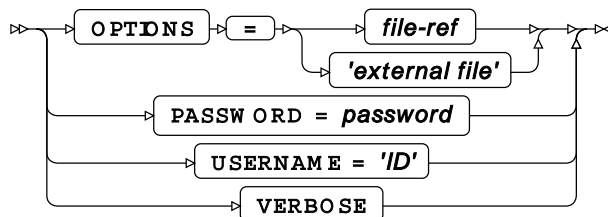
- *PROC HADOOP* [↗](#) (page 2356)
- *HDFS* [↗](#) (page 2356)
- *MAPREDUCE* [↗](#) (page 2357)
- *PIG* [↗](#) (page 2358)

PROC HADOOP

Accesses Hadoop through WPS.

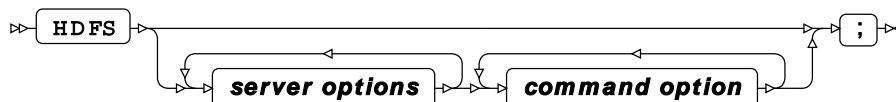


server option

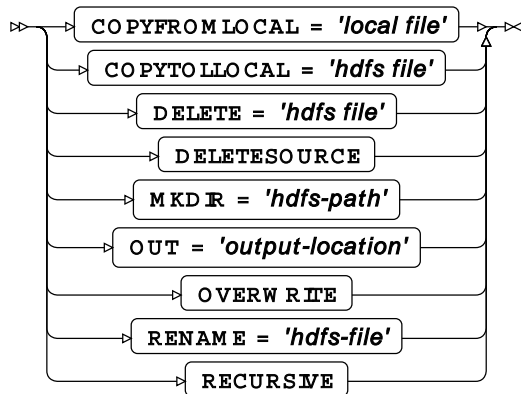


HDFS

Specifies the Hadoop distributed file system to use.

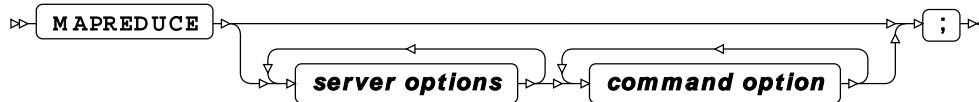


command option

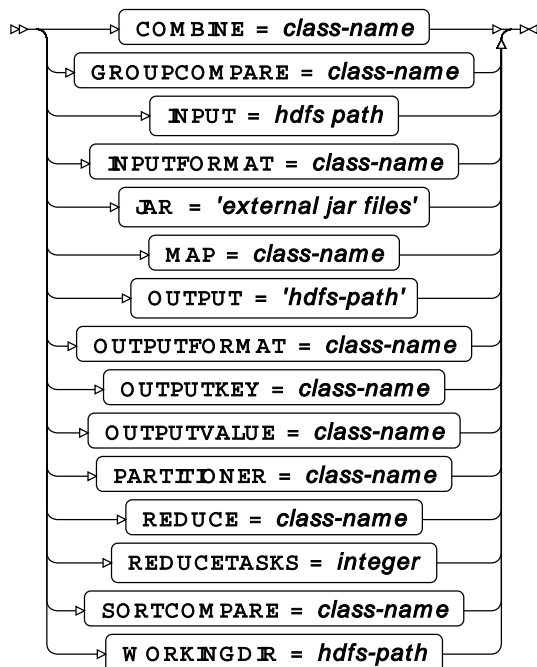


MAPREDUCE

Launches MapReduce jobs.

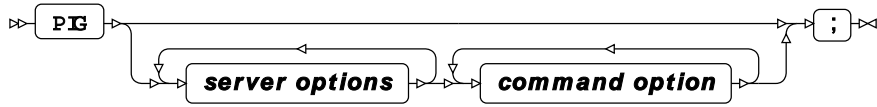


command option

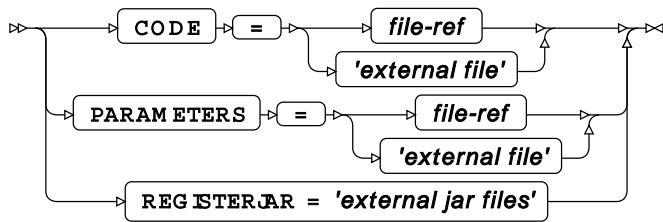


PIG

Enables external files to be submitted to a cluster.



command option



HTTP procedure

Retrieves and updates data from Uniform Resource Locators (URLs) over the Hypertext Transfer Protocol (HTTP).

How to use the HTTP procedure ↗	2358
HTTP procedure reference ↗	2359
Enables data to be retrieved and updated from Uniform Resource Locators (URLs) over the Hypertext Transfer Protocol (HTTP).	

How to use the HTTP procedure

In this example, a trace is requested on the transmitted message so that any changes made by the server, or other servers can be analysed. A HTTP response status code is written to the log to indicate whether the message has been successfully received or rejected by the server.

```

PROC HTTP METHOD="TRACE" URL="http://www.worldprogramming.com/home"
CT="text/html; charset='ISO-8859-4'" HEADEROUT="C:\data\web\headertrace.txt" ;
RUN;
  
```

This produces the following file output:

```
Keep-Alive:timeout=5, max=100
null:HTTP/1.1 302 Found
Server:Apache
Connection:Keep-Alive
Content-Length:221
Date:Thu, 21 Jun 2018 13:51:02 GMT
Content-Type:text/html; charset=iso-8859-1
Location:https://www.worldprogramming.com/home
```

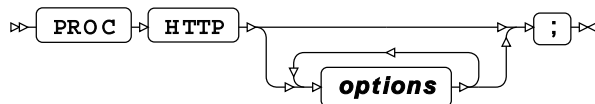
HTTP procedure reference

Enables data to be retrieved and updated from Uniform Resource Locators (URLs) over the Hypertext Transfer Protocol (HTTP).

PROC HTTP 2359
 Enables data to be retrieved and updated from Uniform Resource Locators (URLs) over the Hypertext Transfer Protocol (HTTP). A HTTP response status code is written to the log to indicate whether the message has been successfully received or rejected by the server.

PROC HTTP

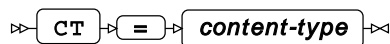
Enables data to be retrieved and updated from Uniform Resource Locators (URLs) over the Hypertext Transfer Protocol (HTTP). A HTTP response status code is written to the log to indicate whether the message has been successfully received or rejected by the server.



Options

The following options are available with the `PROC HTTP` statement

CT



Type: String

Specifies a content-type header field . For example:

```
CT = "text/html; charset='ISO-8859-4'"
```

HEADERIN

⇨ **HEADERIN** ⇨ = ⇨ *file-reference* ⇨

Type: String

Specifies the name of the file (including the pathname) that contains header information to be exported to a Web site. The Web site is only successfully updated if the necessary permissions are in place. For example:

```
PROC HTTP URL='https://www.worldprogramming.com/home' HEADERIN='C:\data\web
\headerin.txt';
RUN;
```

HEADEROUT

⇨ **HEADEROUT** ⇨ = ⇨ *file-reference* ⇨

Type: String

Specifies the name of a file (including the pathname) into which the HTTP header information from a Web site can be saved.

IN

⇨ **IN** ⇨ = ⇨ *file-reference* ⇨

Type: String

Specifies the name of the file (including a pathname) that contains a page structure that can be written to a Web site.

METHOD

⇨ **METHOD** ⇨ = ⇨ *http-method* ⇨

Type: String

Specifies a request method for processing intended internet data from a client to a server. The following methods are available:

GET

Requests data from a specified Web site. For example:

```
PROC HTTP METHOD="GET" URL="http://www.worldprogramming.com/home"
CT="text/html; charset='ISO-8859-4'" HEADEROUT="C:\data\web
\headerout.txt";
```

This produces the following file output:

```
Keep-Alive:timeout=5, max=100
null:HTTP/1.1 302 Found
Server:Apache
Connection:Keep-Alive
Content-Length:221
Date:Thu, 28 Jun 2018 10:53:20 GMT
Content-Type:text/html; charset=iso-8859-1
Location:https://www.worldprogramming.com/home
```

HEAD

Requests data from a specified Web site, but does not return the response body. Use this to check a site exists before downloading a large body of text.

POST

Transfers data to a specified Web site to create a resource, or to perform an update to an existing resource. Multiple `POST` requests to the same Web site will have side effects of creating multiple resources of the same information.

PUT

Transfers data to a specified Web site to create a resource, or to perform an update to an existing resource. Similar to `POST`, however, if you use a `PUT` requests in multiple sessions, it has the same result each time because it replaces the current content with the new information. If a resource does not exist, it creates it.

OPTIONS

Specifies the HTTP methods that the target server supports in the Allow statement, or can interrogate the whole Web site.

DELETE

Deletes the specified resource (if permission has been given to do so).

TRACE

Retransmits the message back from the server so that any changes made by the server, or other servers can be analysed. The message is saved in the file specified by `HEADEROUT`.

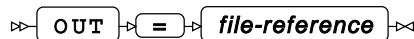
CONNECT

Enables a two-way transparent TCP/IP connection. It is mainly used to link to a Transport Layer Security (TLS) protocol, formally known as a Secure Socket Layer (SSL). All contemporary browsers support this protocol. This method is mainly designed for requests to a proxy. Although the server receiving the request might accept it, most servers do not include this feature.

PATCH

Similar to `PUT`, but modifies and existing HTTP resources instead of replacing a entire resource. The `OPTIONS` method should be used first to establish whether this method is available from the server.

OUT



Type: String

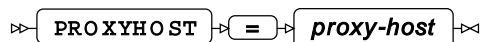
Specifies the `filepath` and `filename` where an output from a Web site of a page structure can be written to. The text `filename` can be named to suit your convention. For example:

```
PROC HTTP URL="https://www.worldprogramming.com/home"  
OUT="C:\data\web\webout.txt";  
RUN;
```

This produces the following file output:

```
<!DOCTYPE html>  
<html prefix="og: http://ogp.me/ns#" lang="en-GB" dir="ltr" id="modernizrcom"  
class="no-js pagewidth320 pagewidth320orbelow pagewidthbelow768">  
<head>  
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
  <meta charset="utf-8">
```

PROXYHOST

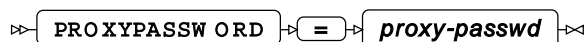


Type: String

Specifies a proxy server that is taking over the tasks for another server. A computer containing the proxy server is known as a proxy server host. For example:

```
PROXYHOST="gateway/genie"
```

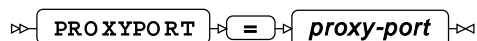
PROXYPASSWORD



Type: String

Specifies a password allowing access to the proxy server.

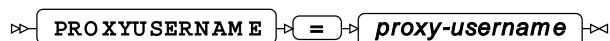
PROXYPORT



Type: Numeric

Specifies a port number of the computer hosting the proxy server.

PROXYUSERNAME



Type: String

Specifies a user name allowing access to the proxy server.

URL

URL = resource-address

Type: String

Specifies an address on the World Wide Web.

WEBDOMAIN

WEBDOMAIN = ntlm-auth-domain

Type: String

Specifies a configured identification string by an authority enabling the user to perform a variety of tasks within the internet.

WEBPASSWORD

WEBPASSWORD = user-passwd

Type: String

Specifies a password enabling access to the internet.

WEBUSERNAME

WEBUSERNAME = user-name

Type: String

Specifies a user name enabling access to the internet.

IMPORT procedure

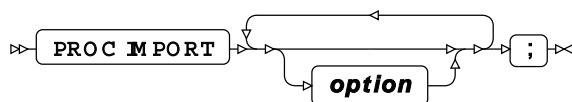
Supported statements

- *PROC IMPORT* [↗](#) (page 2364)
- *DATABASE* [↗](#) (page 2365)
- *DATAROW* [↗](#) (page 2366)
- *DBLIBOPTS* [↗](#) (page 2366)
- *DBPASSWORD* [↗](#) (page 2366)

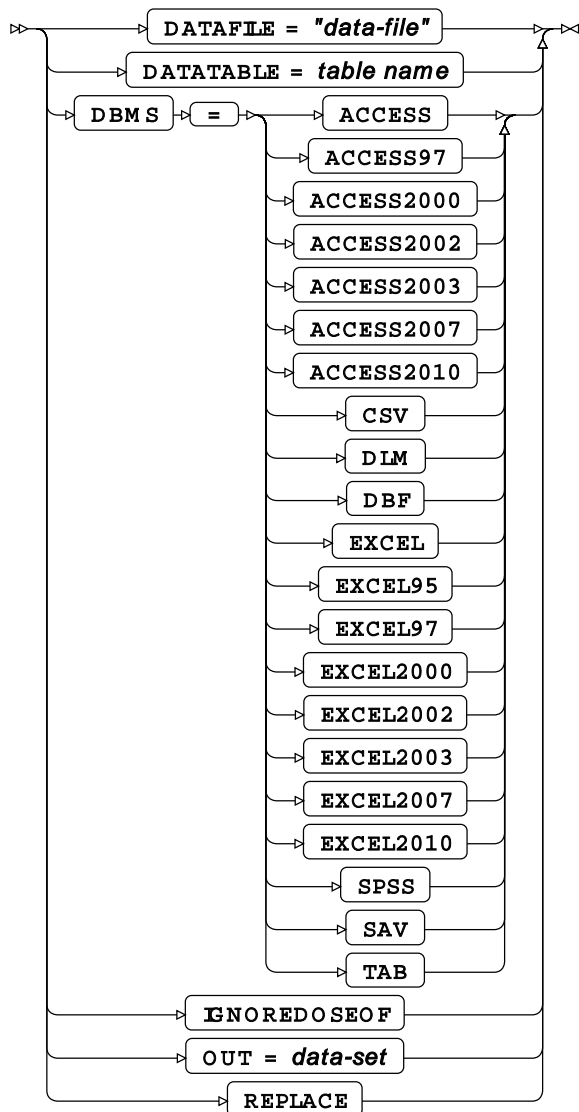
- *DELIMITER* [↗](#) (page 2366)
- *GETDELETED* [↗](#) (page 2366)
- *GETNAMES* [↗](#) (page 2367)
- *GUESSINGROWS* [↗](#) (page 2367)
- *MEMOSIZE* [↗](#) (page 2367)
- *MIXED* [↗](#) (page 2367)
- *MSENGINE* [↗](#) (page 2367)
- *PASSWORD* [↗](#) (page 2368)
- *RANGE* [↗](#) (page 2368)
- *SCANMEMO* [↗](#) (page 2368)
- *SCANTEXT* [↗](#) (page 2368)
- *SCANTIME* [↗](#) (page 2368)
- *SHEET* [↗](#) (page 2369)
- *TEXTSIZE* [↗](#) (page 2369)
- *USEDATE* [↗](#) (page 2369)
- *USER* [↗](#) (page 2369)
- *WGDB* [↗](#) (page 2369)

PROC IMPORT

Imports from an external data source into a WPS dataset.

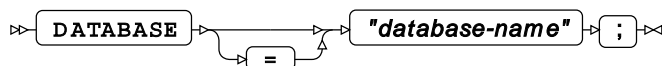


option



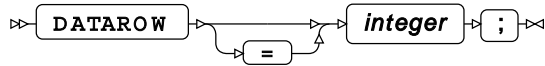
DATABASE

Specifies the database only when **DBMS = MS Access**.



DATAROW

In text files, specifies the line to start getting data from.



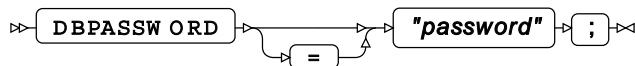
DBLIBOPTS

Specifies database-specific connection options.



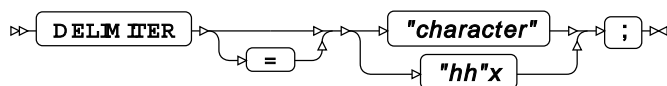
DBPASSWORD

Specifies the database password.



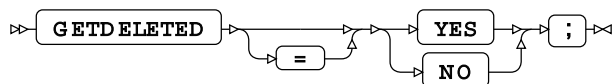
DELIMITER

Specifies the delimiter character (comma, tab).



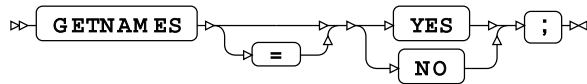
GETDELETED

The GETDELETED option is not supported.



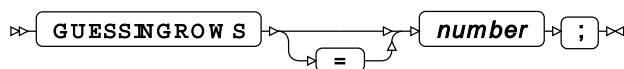
GETNAMES

Specifies whether a line or row which contains field names.



GUESSINGROWS

In text files, specifies how many rows (lines) should be used to work out what the field types are.



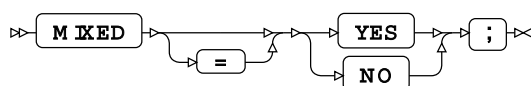
MEMOSIZE

Enters the size of a memo field that the system creates for when it processes a database memo field.



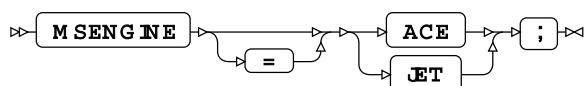
MIXED

Specifies whether to allow both fixed and random effects in analysis.



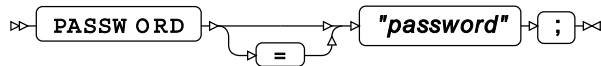
MSENGINE

Selects the Microsoft engine.



PASSWORD

Specifies the password for the data being imported.



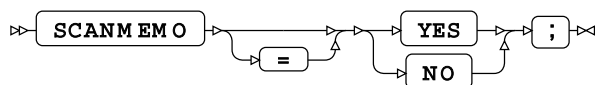
RANGE

Specifies a Microsoft Excel spreadsheet range.



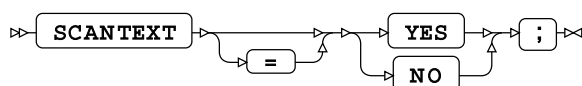
SCANMEMO

Specifies whether `Memo` should be included.



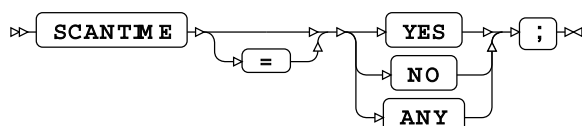
SCANTEXT

Specifies whether `Text` should be included.



SCANTIME

Specifies whether `Time` should be included.



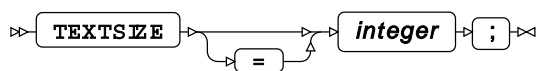
SHEET

Specifies a spreadsheet name.



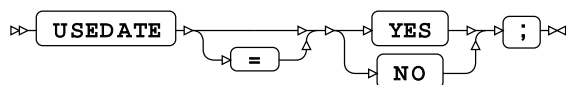
TEXTSIZE

Specifies the size of a text field to use for database text fields.



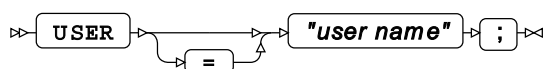
USEDATE

Specifies whether to use current date.



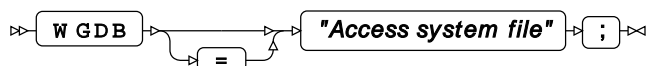
USER

Specifies a user name.



WGDB

Specifies an Access System File name (Working Group Data Base).



JSON Procedure

Enables you to write datasets to a JSON-format file.

JSON (JavaScript Object Notation) syntax is a text-based data format designed to be readable and easy for machines to generate for processing. The syntax uses the following structures:

- A *name–value* pair, for example, a variable name and variable value in an observation from a WPS dataset.
- An *object*. An unordered collection of name–value pairs, for example an observation from a WPS dataset. Names must be unique in each object, and must be a string. String values for both names and values must be formatted to replace any special characters with the JSON *escape sequence*.
- An *array*. An ordered list of values, for example the values in an observation from a WPS dataset.

Special characters in a strings

Some special characters and white space characters in JSON string values must be represented by an *escape sequence*. An escape sequence consists of a backslash (\) and one or more characters to identify the required special character:

Required character	Escape sequence
backslash	\\
quotation marks	\"
forward slash	\/
backspace	\b
form feed	\f
newline	\n
carriage return	\r
horizontal tab	\t
Hexadecimal value	\uvalue

When using the `EXPORT` statement to output a dataset, use the `SCAN` option to correctly format string variables. See [EXPORT](#) (page 2378) for more information.

When using the `WRITE` statement to create JSON data, either write the correctly-formatted string as part of the `VALUE` option, or use the `SCAN` option of the `VALUE` option to correctly format string variables during output. See [WRITE](#) (page 2382) for more information.

Example ↗	2371
These examples shows how to create a JSON-format file.	
JSON procedure reference ↗	2374
Describes the syntax and options for <code>PROC JSON</code> and its contained statements.	

Example

These examples shows how to create a JSON-format file.

The examples are available in `14.0-json-procedure.sas` in the samples distributed with WPS Analytics. You might need to alter the file locations to be able to create the output from the procedure.

Sample Input dataset

The following shows the input dataset that will be written to the JSON file.

```
DATA 'prem17_18'n;
  INFILE CARDS dlm='#';
  FORMAT Home Away $18.;
  FORMAT HS AS WORDS.;
  INFORMAT PlayedOn YYMMDD.;
  FORMAT PlayedOn DATE11.;
  INPUT Home Away HS AS PlayedOn;
  CARDS;
Liverpool      # Manchester City      # 4 # 3 # 2018-01-14
Liverpool      # Manchester United # 0 # 0 # 2017-10-14
Liverpool      # Tottenham Hotspur # 2 # 2 # 2018-02-03
Manchester City # Liverpool          # 5 # 0 # 2017-09-09
Manchester City # Manchester United # 2 # 3 # 2018-04-07
Manchester City # Tottenham Hotspur # 4 # 1 # 2017-12-16
Manchester United # Liverpool          # 2 # 1 # 2018-01-10
Manchester United # Manchester City    # 1 # 2 # 2017-12-09
Manchester United # Tottenham Hotspur # 1 # 0 # 2017-10-24
Tottenham Hotspur # Liverpool          # 4 # 1 # 2017-10-24
Tottenham Hotspur # Manchester City    # 1 # 3 # 2018-04-14
Tottenham Hotspur # Manchester United # 2 # 0 # 2018-01-30
;
RUN;
```

A dataset labelled *prem17_18* is created in the `WORK` folder when the `DATA` step is run. The dataset viewed in Workbench displays the home team score (*HS*) and away team score (*AS*) using the `WORDS.` format.

Basic example

This example outputs a dataset.

The following program reads the sample dataset, referred to using `_LAST_` and outputs the content a JSON-formatted file called `c:\temp\json\results.json`. The `PRETTY` option of the `PROC JSON` statement is used to create a readable output.

```
PROC JSON OUT='c:\temp\json\results.json' PRETTY;
  EXPORT _LAST_;
RUN;
```

Part of the output dataset is shown below:

```
{
  "WPSJSONExport": "1.0 PRETTY",
  "prem17_18": [
    {
      "Home": "Liverpool",
      "Away": "Manchester City",
      "HS": 4,
      "AS": 3,
      "PlayedOn": "14-JAN-2018"
    },
    {
      "Home": "Liverpool",
      "Away": "Manchester United",
      "HS": 0,
      "AS": 0,
      "PlayedOn": "14-OCT-2017"
    },
    ...
    {
      "Home": "Tottenham Hotspur",
      "Away": "Manchester City",
      "HS": 1,
      "AS": 3,
      "PlayedOn": "14-APR-2018"
    },
    {
      "Home": "Tottenham Hotspur",
      "Away": "Manchester United",
      "HS": 2,
      "AS": 0,
      "PlayedOn": "30-JAN-2018"
    }
  ]
}
```

The file is output as an object of name–value pairs. The first entry in the output (WPSJSONExport) is metadata describing the PROC JSON and EXPORT options specified. The dataset name label is the name of the dataset in the WORK folder. The value is an array of objects, where each object is an observation from the dataset.

Example – format output of multiple datasets

This example uses the WRITE statement to create custom JSON output.

The following program reads the sample dataset and outputs selected observations to a JSON-formatted file called c:\temp\json\results-by-division.json.

The PRETTY option of the PROC JSON statement is used to create a readable output. The NOSASTAGS option of the PROC JSON statement is used to prevent multiple versions of the metadata *name–value* pair being output.

- The `WRITE OPEN` statements create the nested object structures to contain the dataset content.
- Each `EXPORT` statement selects part of the sample input dataset based on the name of the *home* team and outputs an array of observations.
- Specifying `NOSASTAGS` means the table name cannot be created as part of the `EXPORT` statement.
- A separate `WRITE VALUES` statement is required to create the dataset label as the *name* in the name–value pair, where the *value* contains the output of the `EXPORT` statement.

```
PROC JSON OUT = 'c:\temp\json\results-by-division.json' PRETTY NOSASTAGS;
  WRITE OPEN OBJECT;
    WRITE VALUES 'premier league';
    WRITE OPEN OBJECT;
      WRITE VALUES 'liverpool';
      EXPORT _LAST_ (WHERE=(HOME='Liverpool'));
      WRITE VALUES 'Manchester City';
      EXPORT _LAST_ (WHERE=(HOME='Manchester City')) /FMTNUMERIC;
    WRITE CLOSE;
  WRITE CLOSE;
RUN;
```

Part of the output dataset is shown below:

```
{
  "premier league": {
    "liverpool": [
      {
        "Home": "Liverpool",
        "Away": "Manchester City",
        "HS": 4,
        "AS": 3,
        "PlayedOn": "14-JAN-2018"
      },
      ...

      {
        "Home": "Liverpool",
        "Away": "Tottenham Hotspur",
        "HS": 2,
        "AS": 2,
        "PlayedOn": "03-FEB-2018"
      }
    ],
    "Manchester City": [
      {
        "Home": "Manchester City",
        "Away": "Liverpool",
        "HS": "five",
        "AS": "zero",
        "PlayedOn": "09-SEP-2017"
      },
      ...

      {
        "Home": "Manchester City",
        "Away": "Tottenham Hotspur",
        "HS": "four",
```

```

    "AS": "one",
    "PlayedOn": "16-DEC-2017"
  }
]
}
}

```

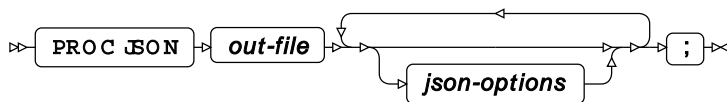
JSON procedure reference

Describes the syntax and options for `PROC JSON` and its contained statements.

<code>PROC JSON</code> ↗	2374
Enables data to be output in JavaScript Object Notation (JSON).	
<code>EXPORT</code> ↗	2378
Enables the output of a WPS dataset.	
<code>WRITE</code> ↗	2382
Enables the creation of free-form output from JSON structures.	

PROC JSON

Enables data to be output in JavaScript Object Notation (JSON).



The `PROC JSON` statement is used to specify the file in which JSON-format data is output. You cannot append data to an existing JSON-format data file. If `out-file` exists, it is overwritten by the procedure.

The `PROC JSON` statement does not output any data. Datasets are output using the `EXPORT` statement; custom JSON-format data is output using the `WRITE` statement.

Options specified on the `PROC JSON` statement apply to all contained statements in the procedure and can be used to specify the general behaviour when outputting data. Where the same option exists on either the `EXPORT` or `WRITE` statements, you can change the behavior for that statement.

For example, you can specify that in general output numeric values are unformatted:

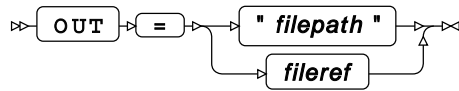
```
PROC JSON 'C:temp\json\output.json' NOFMTNUMERIC;
```

And for one output dataset, specify that numeric values are formatted:

```
EXPORT DATA='myDATASET' / FMTNUMERIC;
```

out-file

Specifies the location of the output file to contain the JSON-format data.



The location can be specified as either a relative or a full path to the file or by using file reference.

filepath

A string, in quotation marks, containing the relative or absolute location of the JSON output file.

fileref

A file reference created using the `FILENAME` global statement.

PROC JSON options

The following options are available with the `PROC JSON` statement.

FMTCHARACTER

Specifies that formatted strings are output from character variables that have a format applied.



To output the raw character values rather than formatted character variables, specify `NOFMTCHARACTER`.

FMTDATETIME

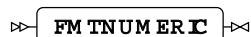
Specifies that formatted strings are output from numeric variables that have a date, datetime or time format applied. This is the default.



To output the raw numeric values rather than formatted date and time variables, specify `NOFMTDATETIME`.

FMTNUMERIC

Specifies that formatted values are output from numeric variables that have a format applied.



The output JSON type is determined by the SAS language format applied. For example the `W.D.` format produces numeric values; the `WORDS.` format produces character values. To output the raw values rather than formatted numeric variables, specify `NOFMTNUMERIC`.

The `FMTNUMERIC` option does not affect numeric variables with date and time formats applied.

KEYS

Specifies that the dataset label and variable names in an exported dataset are output. This is the default.

The icon for the `KEYS` option, consisting of the word `KEYS` in a rounded rectangle with arrowheads on the left and right sides.

When specified, datasets are output as a *name–value* pair in an object. The *name* is the dataset label, the *value* is an array of objects, where each object is an observation from the dataset.

NOFMTCHARACTER

Specifies that raw values are output from character variables that have a format applied. This is the default.

The icon for the `NOFMTCHARACTER` option, consisting of the text `NOFMTCHARACTER` in a rounded rectangle with arrowheads on the left and right sides.

To output character variables with applied formats, specify `FMTCHARACTER`.

NOFMTDATETIME

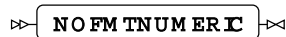
Specifies that raw values are output from numeric variables that have a date, datetime or time format applied.

The icon for the `NOFMTDATETIME` option, consisting of the text `NOFMTDATETIME` in a rounded rectangle with arrowheads on the left and right sides.

To output numeric variables with applied formats, specify `FMTDATETIME`.

NOFMTNUMERIC

Specifies that the raw values are output from numeric variables that have a format applied. This is the default.

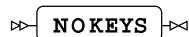
The icon for the `NOFMTNUMERIC` option, consisting of the text `NOFMTNUMERIC` in a rounded rectangle with arrowheads on the left and right sides.

To output numeric variables with date and time formats applied are as formatted strings, specify `FMTDATETIME`.

The `NOFMTNUMERIC` option does not affect numeric variables with date and time formats applied

NOKEYS

Specifies that only the variable values in an exported dataset are output.

The icon for the `NOKEYS` option, consisting of the text `NOKEYS` in a rounded rectangle with arrowheads on the left and right sides.

When specified, datasets are output as a *name–value* pairs in an object. The *name* is the dataset label, the *value* is an array of arrays, where each nested array contains a list of variable values for one observation in the dataset.

NOPRETTY

Specifies that the JSON-format data is output to the *out-file* in an unformatted, compressed form. This is the default.

The icon for the `NOPRETTY` option, consisting of the text `NOPRETTY` in a rounded rectangle with arrowheads on the left and right sides.

When specified, a valid JSON structure is output to *out-file* that might be smaller in size than similar output using the `PRETTY` option. Subsequent processing of the *out-file* might therefore be faster because of a smaller file size.

NOSASTAGS

Specifies that export metadata and dataset variable labels are not output.

➤ `NOSASTAGS` ➤

NOSCAN

Specifies character variables are not processed to convert special characters to their *escape sequence* before the variable is output.

➤ `NOSCAN` ➤

If the character variable is not correctly formatted before output, an invalid JSON-format file might be created.

To convert special characters to their *escape sequence* before output to the *out-file*, specify `SCAN`.

NOTRIMBLANKS

Specifies that leading and trailing blank spaces are not removed from values output.

➤ `NOTRIMBLANKS` ➤

When specified the size of *out-file* on disk might increase because of lengths or formats defined for character variables in the exported dataset.

PRETTY

Specifies that the *out-file* file is formatted to make the JSON content easier to read and understand.

➤ `PRETTY` ➤

When specified, the file content has the following formatting applied:

- The start (`{`) and end (`}`) markers of an array are output on separate lines.
- The start (`[`) and end (`]`) markers of a list are output on separate lines.
- Each entry in an array or list is indented to indicate the format hierarchy.
- Each name–value pair in an array is output on a separate line.
- Each value in a list is output on a separate line.

Using the `PRETTY` option might increase the size of the specified *out-file* on disk because the formatting includes formatting.

SASTAGS

Specifies that metadata for the datasets is output. This is the default.

➤ SASTAGS ➤

Exported metadata is a list of options specified on the `PROC JSON` statement and the `EXPORT` statement used to output a dataset. The `SASTAGS` option also controls the output of the dataset label name.

This metadata is output into a variable named `WPSJSONExport`. There is one export metadata value for each dataset output.

SCAN

Specifies that special characters in a string value are converted to their *escape sequence* and output. This is the default.

➤ SCAN ➤

To stop an *escape sequence* being created, for example if the value has been correctly formatted before being output to the *out-file*, specify `NOSCAN`.

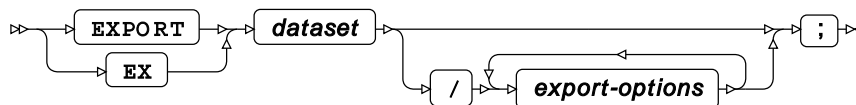
TRIMBLANKS

Specifies that leading and trailing blank spaces are removed from string values before being output. This is the default.

➤ TRIMBLANKS ➤

EXPORT

Enables the output of a WPS dataset.



A dataset is, by default, output as an array of objects. Each object represents an observation in the dataset, and contains a *name–value* pair, for each variable name and variable value in the observation.

An observation can be output as a list of values by specifying the `NOKEYS` option. The default label for the dataset is the name specified in *dataset*, which can be changed using the `TABLENAME` option.

Multiple `EXPORT` statements can be used in a JSON procedure. Metadata is created for each `EXPORT` statement, but the identifying Name in the *name–value* pair is same each time metadata is output. If multiple statements are used, specify `NOSASTAGS` on the `PROC JSON` statement, or each `EXPORT` statement after the first to prevent output of multiple metadata information.

dataset

Specifies the dataset to be output to *out-file* specified on the `PROC JSON` statement.

EXPORT options

The following options are available with the `EXPORT` statement.

FMTCHARACTER

Specifies that formatted strings are output from character variables in *dataset* that have a format applied.

➤ **FMTCHARACTER** ➤

To output the raw character values rather than formatted character variables, specify `NOFMTCHARACTER`.

FMTDATETIME

Specifies that formatted strings are output from numeric variables in *dataset* that have a date, datetime or time format applied.

➤ **FMTDATETIME** ➤

To output the raw numeric values rather than formatted date and time variables, specify `NOFMTDATETIME`.

FMTNUMERIC

Specifies that formatted values are output from numeric variables in *dataset* that have a format applied.

➤ **FMTNUMERIC** ➤

The output JSON type is determined by the SAS language format applied. For example the `W.D.` format produces numeric values; the `WORDS.` format produces character values. To output the raw values rather than formatted numeric variables, specify `NOFMTNUMERIC`.

The `FMTNUMERIC` option does not affect numeric variables with date and time formats applied.

KEYS

Specifies that the dataset label and variable names in an exported dataset are output.

➤ **KEYS** ➤

When specified, datasets are output as a *name–value* pair in an object. The *name* is the dataset label, the *value* is an array of objects, where each object is an observation from the dataset.

NOFMTCHARACTER

Specifies that raw values are output from character variables in *dataset* that have a format applied. This is the default

➤ **NO FMT CHARACTER** ➤

To output character variables with applied formats, specify `FMTCHARACTER`.

NOFMTDATETIME

Specifies that raw values are output from numeric variables in *dataset* that have a date, datetime or time format applied.

➤ **NO FMT DATETIME** ➤

To output numeric variables with applied formats, specify `FMTDATETIME`.

NOFMTNUMERIC

Specifies that the raw values are output from numeric variables in *dataset* that have a format applied.

➤ **NO FMT NUMERIC** ➤

To output numeric variables with applied formats, specify `FMTNUMERIC`.

NOKEYS

Specifies that only the variable value in an exported dataset is output.

➤ **NOKEYS** ➤

When specified, the dataset is output as a *name–value* pair in an object. The *name* is the dataset label or `TABLENAME`. The *value* is an array of arrays, where each nested array contains a list of variable values for one observation in the dataset.

NOSASTAGS

Specifies that metadata for *dataset* is not output.

➤ **NO SASTAGS** ➤

If specified, the dataset name or any alternative label specified using the `TABLENAME` keyword is not output. If this would result in invalid JSON-format, the dataset is not output.

NOSCAN

Specifies character variables in *dataset* are not processed to convert special characters to their *escape sequence* before the variable is output.

➤ **NO SCAN** ➤

If the character variable is not correctly formatted before output, an invalid JSON-format file might be created.

To convert special characters to their *escape sequence* before output to the *out-file*, specify `SCAN`.

NOTRIMBLANKS

Specifies that leading and trailing blank spaces are not removed from character variables in *dataset* when output.

➤ **NO TRIM BLANKS** ➤

When specified the size of *out-file* on disk might increase because of lengths or formats defined for character variables in the exported dataset.

SASTAGS

Specifies that metadata for *dataset* is output.

➤ **SASTAGS** ➤

Exported metadata is a list of options specified on the `PROC JSON` statement and options specified on the `EXPORT` statement. Metadata is output into a variable named *WPSJSONExport*.

The `SASTAGS` option also controls the output of the dataset label name specified using the `TABLENAME` option.

SCAN

Specifies that special characters in character variables in *dataset* are converted to their *escape sequence* and output.

➤ **SCAN** ➤

To stop an *escape sequence* being created, for example if the value has been correctly formatted in the dataset, specify `NOSCAN`.

TABLENAME

Specifies a different name for specified *dataset*.

➤ **TABLENAME = *label*** ➤

The specified *label* is not output if `NOSASTAGS` is specified. The label must be entered in quotation marks if it contains spaces, and must be formatted to replace any special characters with the JSON *escape sequence*.

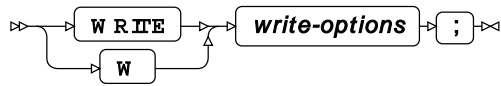
TRIMBLANKS

Specifies that leading and trailing blank spaces are removed from character variables in *dataset* before being output.

➤ **TRIM BLANKS** ➤

WRITE

Enables the creation of free-form output from JSON structures.



`WRITE` statements are used in a JSON procedure to create a required output structure and content. `WRITE` provides more control over output than is available using the `EXPORT` statement.

`WRITE` statements are used to create the structure of arrays and objects using the `OPEN` option. Values are output in the created structure using the `VALUES` option.

Multiple `WRITE OPEN` statements can be combined to create a structure of nested objects to create a hierarchy in which data can be output. `WRITE CLOSE` enables data to be grouped in the output.

When using the `WRITE OPEN` statements, you should ensure the output structure is correct, for example that arrays and objects are not output as the name in a *name–value* pair.

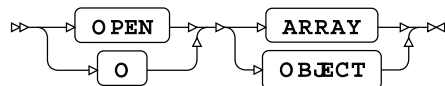
If the procedure step only contains `WRITE VALUES` statements, the output is *name–value* pairs in a JSON object.

WRITE options

The following options are available with the `WRITE` statement.

OPEN

Specifies that a JSON structure is created. Values can be written into the structure using the `WRITE VALUES` statement. Nested structures are created using the `OPEN ARRAY` or `OPEN OBJECT` options.



ARRAY

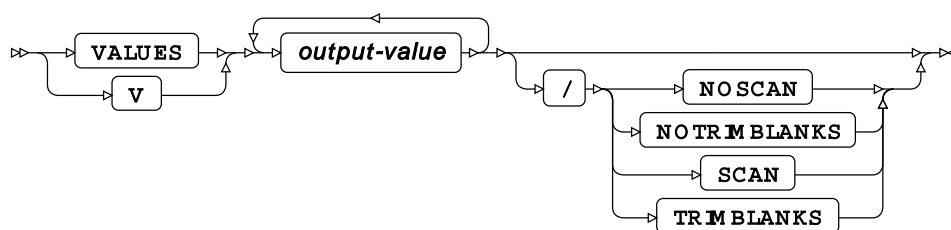
Creates an array – an ordered collection of values or objects.

OBJECT

Creates an object – an unordered collection of name–value pairs.

VALUES

Specifies the output values written to an object or array in the JSON file.



Multiple values can be specified using the same `WRITE VALUES` statement.

If the values are output to an array each value is output as a separate entry, for example:

```
PROC JSON OUT = 'json/array.json';
WRITE OPEN ARRAY;
WRITE VALUES 'hello' 'world';
WRITE CLOSE;
RUN;
```

Outputs the following JSON-structure to the specified file:

```
[ "hello", "world" ]
```

If the values are output to an object, values are treated as *name–value* pairs in the order they are encountered.

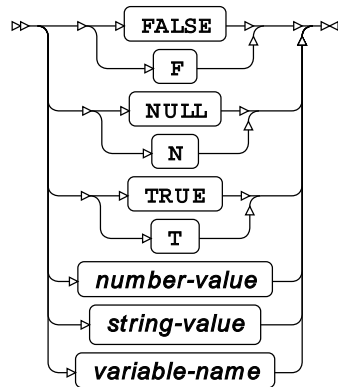
```
PROC JSON OUT = 'json/object.json';
WRITE OPEN OBJECT;
WRITE VALUES 'hello' 'world' 'how' 'boiled';
WRITE CLOSE;
RUN;
```

The output JSON file contains the following object structure:

```
{ "hello": "world", "how": "boiled" }
```

output-value

Specifies one or more values output to a JSON structure.



FALSE

Outputs the value `false`. Cannot be used as the name in a name–value pair of an object.

NULL

Outputs the value `null`. Cannot be used as the name in a name–value pair of an object.

TRUE

Outputs the value `true`. Cannot be used as the name in a name–value pair of an object.

number-value

Outputs a numeric value.

string-value

Outputs a string value. Strings in JSON-format data are output in quotation marks. If *string-value* contains any special characters they should be converted to the equivalent *escape sequence* before output.

string-value must be in quotation marks if the value contains spaces or special characters. If *string-value* is not entered in quotation marks, values are delimited by spaces, for example,

```
PROC JOSN OUT='json/outfile.json';  
  WRITE OPEN ARRAY;  
    WRITE VALUES HELLO WORLD;  
RUN;
```

This creates two entries in the array list: "HELLO " and "WORLD".

If quotation marks are used, values can be entered in either single or double quotation marks. When using single quotes, in addition to replacing special characters with the equivalent escape sequence, apostrophes in text must be preceded by an apostrophe as an escape character.

variable-name

Outputs the content of the specified variable name.

NOSCAN

Specifies that character variables are not processed to convert special characters to their *escape sequence* before the variable is output.

NOTRIMBLANKS

Specifies that leading and trailing blank spaces are not removed from values before being output.

SCAN

Specifies that special characters in a string value are converted to their *escape sequence* and output.

TRIMBLANKS

Specifies that leading and trailing blank spaces are removed from string values before being output.

CLOSE

Specifies the current array or object is closed.



If not specified, all open objects and arrays are automatically closed at the end of the JSON procedure step. Specifying `CLOSE` enables a nested structure of arrays and objects to be created in the JSON output.

Example – manually create a JSON-structure

The following example shows how to create a JSON structure using the `WRITE` statement. The top structure is an object in which defined name–value pairs are output. The first pair consists of a name that is the dataset label, `Dates`, and an array to contain the dataset.

Each observation in the dataset is an object and name–value pairs output before the object is closed. The array containing the observations is then closed as the end of the procedure step.

```
PROC JSON OUT = 'json/output.json' PRETTY;
  WRITE OPEN OBJECT;
    WRITE VALUE 'Dates';
    WRITE OPEN ARRAY;
      WRITE OPEN OBJECT;
        WRITE VALUES 'Date' '2019-01-01';
        WRITE VALUES 'Label' 'New Year's Day';
      WRITE CLOSE;
      WRITE OPEN OBJECT;
        WRITE VALUES 'Date' '2019-01-02';
        WRITE VALUES 'Label' 'Second of January';
      WRITE CLOSE;
    WRITE CLOSE;
  WRITE CLOSE;
RUN;
```

Which outputs the following structure.

```
{
  "Dates": [
    {
      "Date": "2019-01-01",
      "Label": "New Year's Day"
    },
    {
      "Date": "2019-01-02",
      "Label": "Second of January"
    }
  ]
}
```

JAVAINFO procedure

Provides information about the Java virtual machine used by WPS.

How to use the JAVAINFO procedure [↗](#)..... 2386

JAVAINFO procedure reference [↗](#)..... 2387

Provides information about the Java virtual machine used by WPS.

How to use the JAVAINFO procedure

In this example, all Java information is requested. The result is written to the log.

```
PROC JAVAINFO;  
RUN;
```

This produces the following output:

```
java.version = 1.8.0_51  
java.vendor = Oracle Corporation  
java.vendor.url = http://java.oracle.com/  
java.home = C:\Program Files\World Programming\WPS\4\jre  
java.ext.dirs = C:\Program Files\World Programming\WPS\4\jre\lib\ext;C:\WINDOWS\Sun  
\Java\lib\ext  
java.security.policy = <no value>  
javaplugin.version = <no value>  
java.vm.specification.version = 1.8  
java.vm.specification.vendor = Oracle Corporation  
java.vm.specification.name = Java Virtual Machine Specification  
java.vm.version = 25.51-b03  
java.vm.vendor = Oracle Corporation  
java.vm.name = Java HotSpot(TM) 64-Bit Server VM  
java.specification.version = 1.8  
java.specification.vendor = Oracle Corporation  
java.specification.name = Java Platform API Specification  
java.class.version = 52.0  
java.class.path = C:\Program Files\World Programming\WPS\4\jars\wps.jar;C:\Program  
Files\World  
Programming\WPS\4\jars\wpsssh.jar;C:\Program Files\World  
Programming\WPS\4\jars\poi-3.13-20150929.jar;C:\Program Files\World  
Programming\WPS\4\jars\poi-ooxml-3.13-20150929.jar;C:\Program Files\World  
Programming\WPS\4\jars\poi-ooxml-schemas-3.13-20150929.jar;C:\Program Files\World  
Programming\WPS\4\jars\xmlbeans-2.6.0.jar  
JREOPTIONS = ('-Djavax.security.auth.useSubjectCredsOnly=false'  
'-Djava.class.path=!wpshome\jars\wps.jar;!wpshome\jars\wpsssh.jar;!wpshome\jars/  
poi-3.13-2015092  
9.jar;!wpshome\jars\poi-ooxml-3.13-20150929.jar;!wpshome\jars\poi-ooxml-  
schemas-3.13-20150929.ja  
r;!wpshome\jars\xmlbeans-2.6.0.jar' '-Dwps.jre.libjvm=!wpshome\jre\bin\server\  
\jvm.dll'  
'-Djava.security.auth.login.config=!wpshome\jars\jaas.config')  
os.name = Windows NT (unknown)  
os.version = 10.0  
os.arch = amd64  
file.separator = \  
path.separator = ;  
line.separator =  
  
user.name = david.jones  
user.home = C:\Users\david.jones  
user.dir = C:\Users\david.jones\Documents\WPS Workspaces\Workspace1\Samples  
\Java_info
```

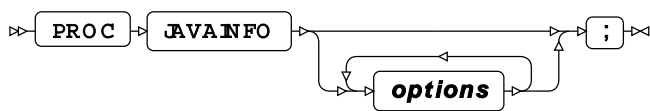

JAVAINFO procedure reference

Provides information about the Java virtual machine used by WPS.

PROC JAVAINFO ↗	2387
Provides information about the Java virtual machine used by WPS.	

PROC JAVAINFO

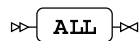
Provides information about the Java virtual machine used by WPS.



Options

The following options are available with the PROC JAVAINFO statement

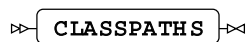
ALL



Type: Keyword

Specifies comprehensive Java information by activating all the other valid options with the exception of `HELP`. This is also the default setting. The results are written to the log.

CLASSPATHS



Type: Keyword

Specifies the Java Virtual Machine or Java compiler environment variable that identifies the paths of user-defined classes and packages. The result is written to the log.

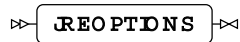
HELP



Type: Keyword

Specifies all valid options. The result is written to the log.

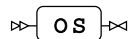
JREOPTIONS



Type: Keyword

Specifies options for the Java runtime environment. The result is written to the log.

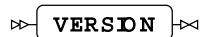
OS



Type: Keyword

Specifies information about the operating system that Java is running on. The result is written to the log.

VERSION



Type: Keyword

Specifies comprehensive information about the Java version used.

In this example, Java version information is requested. The result is written to the log.

```
PROC JAVAINFO VERSION;  
RUN;
```

This produces the following output:

```
java.version = 1.8.0_51  
java.vendor = Oracle Corporation  
java.vendor.url = http://java.oracle.com/  
java.home = C:\Program Files\World Programming\WPS\4\jre  
java.ext.dirs = C:\Program Files\World Programming\WPS\4\jre\lib\ext;  
C:\WINDOWS\Sun\Java\lib\ext  
java.security.policy = <no value>  
javaplugin.version = <no value>  
java.vm.specification.version = 1.8  
java.vm.specification.vendor = Oracle Corporation  
java.vm.specification.name = Java Virtual Machine Specification  
java.vm.version = 25.51-b03  
java.vm.vendor = Oracle Corporation  
java.vm.name = Java HotSpot(TM) 64-Bit Server VM  
java.specification.version = 1.8  
java.specification.vendor = Oracle Corporation  
java.specification.name = Java Platform API Specification  
java.class.version = 52.0
```

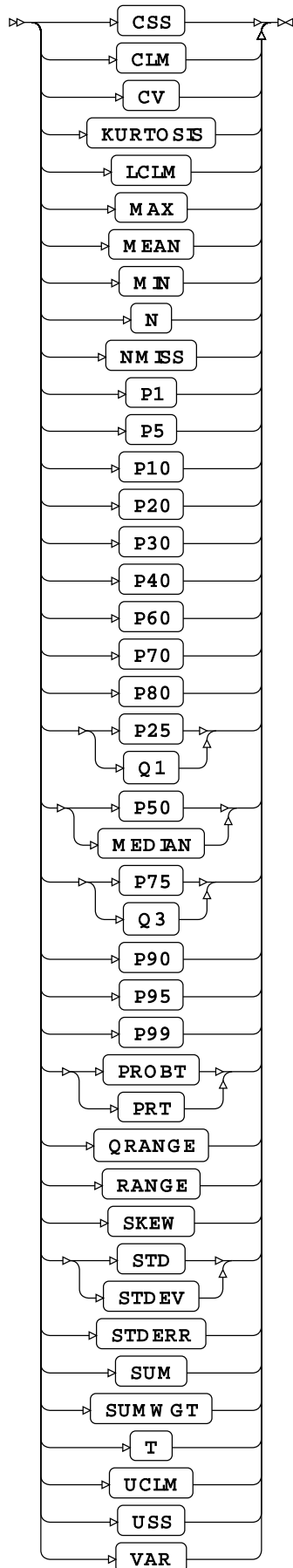
MEANS procedure

Supported statements

- *PROC MEANS* [↗](#) (page 2391)
- *BY* [↗](#) (page 2393)
- *CLASS* [↗](#) (page 2393)
- *FREQ* [↗](#) (page 2393)
- *ID* [↗](#) (page 2394)
- *OUTPUT* [↗](#) (page 2394)
- *TYPES* [↗](#) (page 2395)
- *VAR* [↗](#) (page 2395)
- *WAYS* [↗](#) (page 2395)
- *WEIGHT* [↗](#) (page 2396)
- *WHERE* [↗](#) (page 2396)

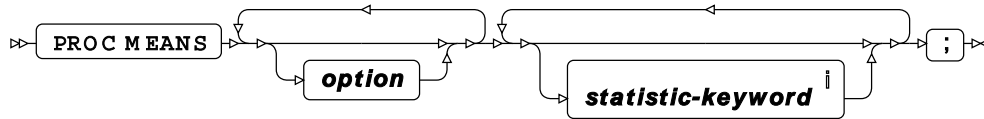
Statistic keywords

The following keywords are used within several statements of this procedure.



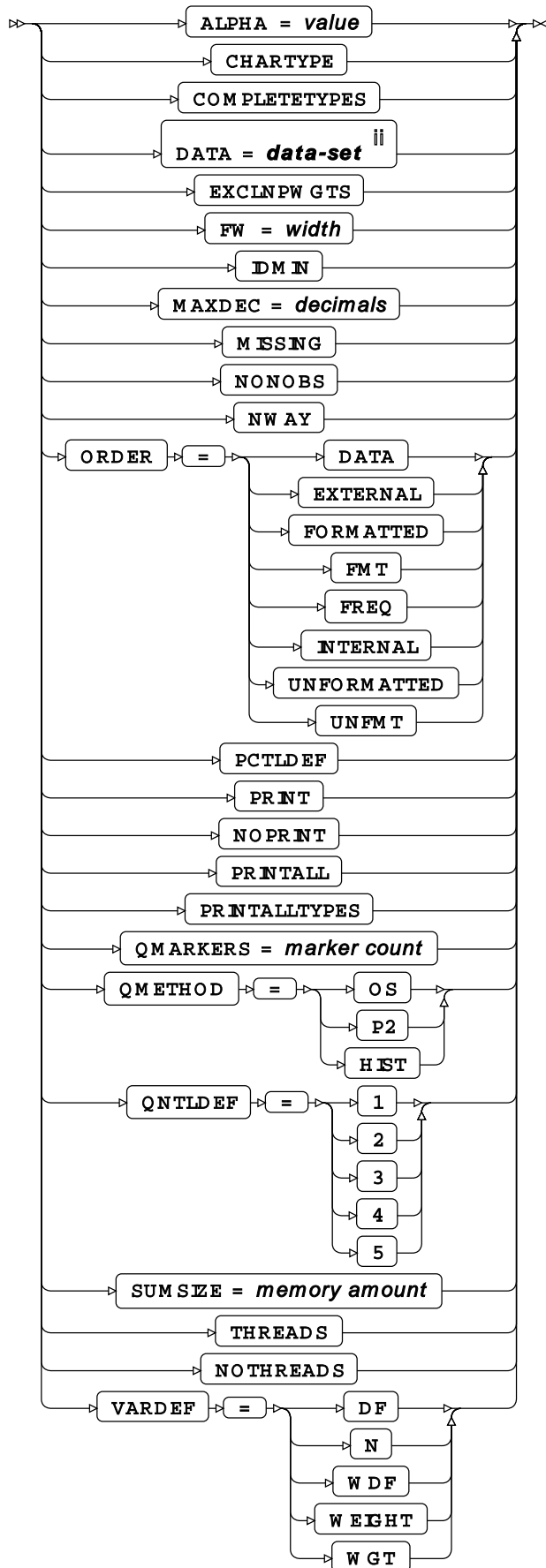
PROC MEANS

Calculates elementary statistics for a dataset.



ⁱ See *Statistic keywords* [↗](#) (page 2389).

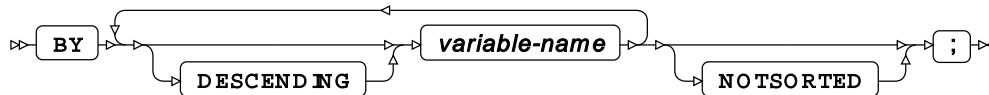
option



ii See *Input dataset* [↗](#) (page 16).

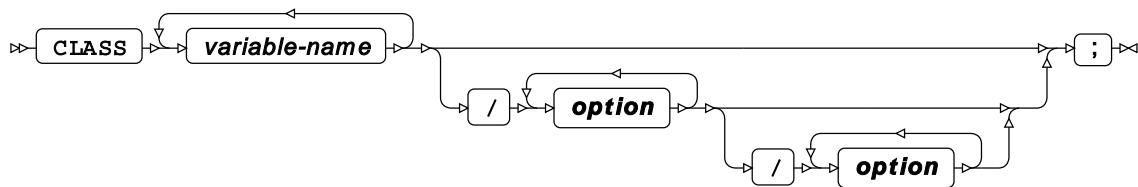
BY

Groups the observation using one or more specified variables.

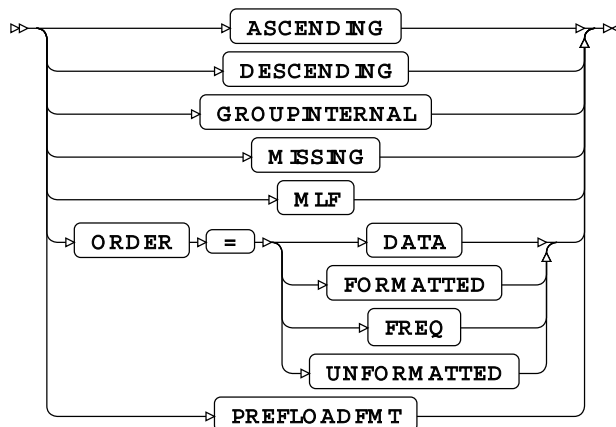


CLASS

Specifies variables (within a BY group), by which observations are to be grouped.

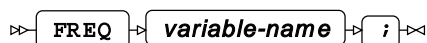


option



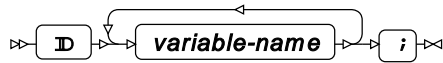
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



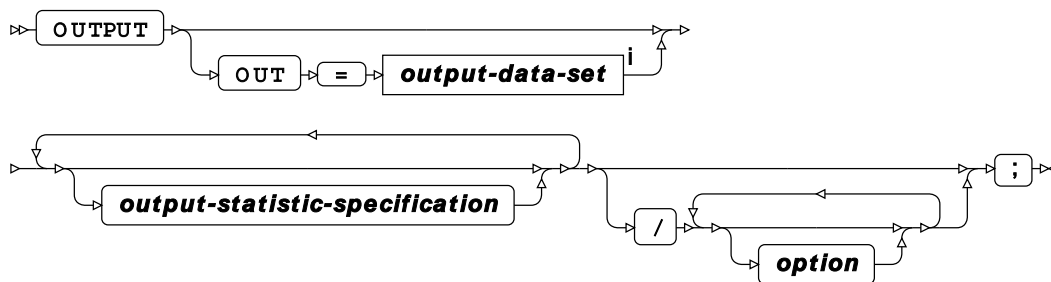
ID

Identifies the relevant observations in the output by using one or more specified variable names.



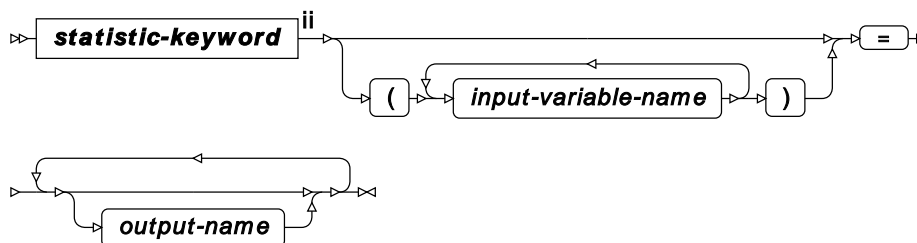
OUTPUT

Creates an output dataset containing data given by one or more statistic keyword specifications.



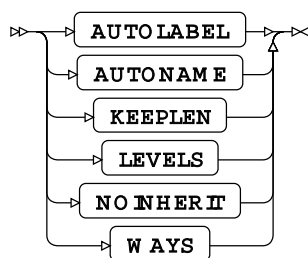
ⁱ See *Output dataset* [↗](#) (page 16).

output-statistic-specification



ⁱⁱ See *Statistic keywords* [↗](#) (page 2389).

option

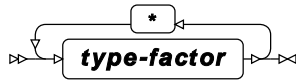


TYPES

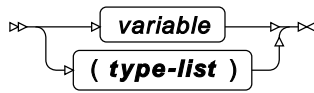
Restricts output to subsets of `CLASS` variables.



type-list

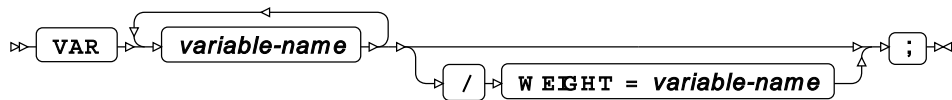


type-factor



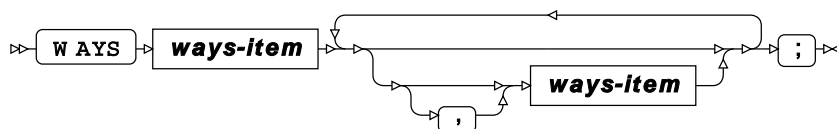
VAR

Specifies variables for which to calculate statistics.

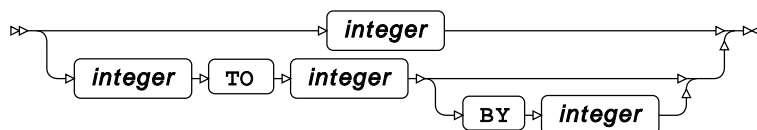


WAYS

Restricts the number of outputs, for example, a one-dimensional table, a two-dimensional table, or both.



ways-item



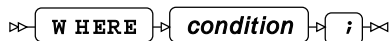
WEIGHT

Specifies a variable giving the weight associated with each observation.



WHERE

Restricts the observations to be processed.



OPTIONS procedure

Returns the values of all system options, or of a specified system option.

The system options are described in detail in the section [System options](#) (page 42).

The procedure also returns information about environment variables. The output is grouped into host system options, portable system options, and environment variables.

Host system options apply only to the system on which the system is running; portable options apply to any system. For example, some system options are specific to z/OS; if any of these were set for a WPS session on z/OS, these would be host options, while any options not specific to z/OS would be portable options.

Examples ↗	2396
Examples of using the procedure.	
OPTIONS procedure reference ↗	2398
Describes the syntax and options for PROC OPTIONS.	

Examples

Examples of using the procedure.

Basic example ↗	2397
In this example, concise information about system options is returned. The result is written to the log.	
Example – getting information for a specified option ↗	2397
In this example, information is returned for a specified system option. The result is written to the log.	

Basic example

In this example, concise information about system options is returned. The result is written to the log.

```
proc options short;
```

This produces the following output:

```
Portable Options:

ALTLOG=  NOAUTOSIGNON  BASEENGINE=WPD  NOBATCHWPDLOCKING  BOMFILE  BUFNO=1
BUFSIZE=0  BUFSIZECMULT=6  BUFSIZEUMULT=1  BYERR  BYLINE  NOCAPS  CARDIMAGE
CENTER  NOCHARCODE  CLEANUP  CMPLIB=  COMAMID=TCP  COMPRESS=NO
CONFIG=( 'C:\Program Files\World Programming\WPS\4\wps.cfg' ) NOCONFIGFONT
CONNECTPERSIST  CONNECTREMOTE=  CONNECTWAIT  NOCONSIDERXLSXCOLWIDTHS  CPORTVER=SAS92

DATASTMTCHK=COREKEYWORDS  DATE  DATESTYLE=DMY  DBSLICEPARM=(THREADED_APPS, 2)
.
.
.

Host Options:

AUTOEXEC=  BOTTOMMARGIN=1CM  CPUCOUNT=8  NODLCREATEDIR  NODMS
EBCDICFMTINFMTBEHAVIOUR=DEFAULT EMAILAUTHDOMAIN=  EMAILAUTHPROTOCOL=NONE
EMAILHOST=localhost  EMAILID=  EMAILMASQUERADEHOST=EMAILPORT=25  EMAILPW=*****
EMAILSTARTTLS=AUTO  EMAILSYS=MAPI  ENCODING=WLATIN1
.
.
.

Environment Variables:

SASAUTOS=('!wps\home\sasmacro')
```

The output contains three sections, for host and portable system options, and for environment variables. The output shown in the example has been abbreviated to save space.

Example – getting information for a specified option

In this example, information is returned for a specified system option. The result is written to the log.

```
PROC OPTIONS OPTION=ERRORS SHORT;
```

This produces the following output:

```
ERRORS=20
```

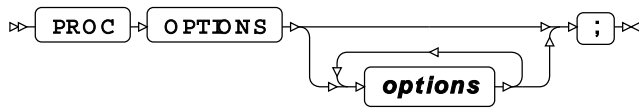
OPTIONS procedure reference

Describes the syntax and options for PROC OPTIONS.

PROC OPTIONS ↗	2398
Enables you to specify the level of detail of the information returned, and to specify a system option for which you want information returned.	

PROC OPTIONS

Enables you to specify the level of detail of the information returned, and to specify a system option for which you want information returned.



You can specify that the information returned is detailed, a summary, concise and so on. You can return information about all system options, or for a specified system option. Some options return information that is grouped into host and portable system options.

PROC OPTIONS with no options is the same as PROC OPTIONS LONG.

If OPTION is not specified, information for all system options is returned. If OPTION is specified, information for the specified system option is returned.

If the DEFINE option is not specified, the information returned includes the name of the system option, the current setting, and a description of the system option. This is the format returned by the LONG option, and is the default. See LONG for more information.

Options

The following options are available with the PROC OPTIONS statement

DEFINE

Returns detailed information about system options.



Type: Keyword

The detailed information includes the name of the system option, the current setting, the group to which it belongs, information about the group, a description of it, and so on.

If OPTION is not specified, lists detailed information for all system options, including the current setting and a description of the system option.

If `OPTION` is specified, lists detailed information about, and the current setting of the specified system option.

The information returned is split into two groups, host system options and portable system options. For each system option, the following information is provided:

- The name of the system option and its current setting.
- The name of the group to which the system option belongs.
- A description of the group.
- A description of the system option.
- The type of system option, this can be:

- `STRING`

This provides the following additional information:

- The maximum number of characters
- Whether the case of the system option's value is retained
- Whether quotation marks have been removed
- Whether the value of the system option required parentheses

- `LONG`

This also provides the range of the values allowed.

- `INTMAX`

This also provides the range of the values allowed.

- `BOOLEAN`

- Where the system option can be set. Some system options are limited in where they can be set. For example, `AUTOEXEC` can only be set in a configuration file or on the command line; `CENTER` can be set using the `OPTIONS` global statement, in a configuration file, or on the command line.
- Whether the system option can be restricted.
- Whether the system option is currently restricted.

A system option can be restricted to prevent it being changed during a WPS session. A system option can be restricted in various ways. See *Restricting system options* [↗](#) (page 42) in *System options* for more information.

EXPAND

Returns expanded symbolic references.



Type: Keyword

For example, the `JREOPTION` system option can be used to specify options for the Java Runtime Environment (JRE) , including class paths, such as:

```
C:\Program Files\World Programming\WPS\4\jars\wps.jar
```

These paths can be specified using a symbolic reference:

```
!wpshome\jars\wps.jar
```

Where `!wpshome` is a symbolic reference for the path `C:\Program Files\World Programming\WPS\4\`.

If you specify this option, symbolic references are expanded.

GROUP

Returns a list of the system options associated with a group, and their settings.

➤ **GROUP** ➤ = ➤ *group-name* ➤

Type: String

System options belong to one or more groups. You can display the group names using the `LISTGROUPS` option. You can then use the group name to list the system option in that group. You can only display groups that are applicable to your operating system.

You can specify more than one group to this option; for example:

```
PROC OPTIONS GROUP ENVDISPLY MACRO
```

This option must be the last option specified, otherwise subsequent options are assumed to be group names.

For example, the group `SORT` contains all the system options associated with sorting. To see all of the system options in this group, you would specify the group name to this option:

```
PROC OPTIONS GROUP=sort;
```

HEXVALUE

Character values for options are displayed as hexadecimal values.

➤ **HEXVALUE** ➤

Type: Keyword

For example, `PROC OPTIONS OPTION=sortpgm;` returns the name of the current sort program. By default this is `WPS`, and by default this is returned as a character value. If you specify this option the name of the sort program is instead returned as a hexadecimal value. For example:

```
proc options option=sortpgm hexadecimal;
```

This returns 575053, where 57 is the hexadecimal representation of the character W, 50 is the hexadecimal representation of the character P, and 53 is the hexadecimal representation of the character S.

HOST

Returns information only for host system options.

» HOST «

Type: Keyword

System options are of two general types, host and portable. By specifying this option, only information about host options is returned. The information is provided in the same format as the EXPAND option.

Environment variables are also returned.

LISTGROUPS

Returns a list of groups.

» LISTGROUPS «

Type: Keyword

System options belong to groups. This option enables you to list all of the group names. You can then use the group name to list all of the system options in that group by specifying it to the GROUP option. The list only contains groups applicable to your operating system.

LISTINSERTAPPEND

Returns a list of system options and environment variables that can be specified with INSERT and APPEND operations.

» LISTINSERTAPPEND «

Type: Keyword

The information returned is split into three groups, host and portable options, and environment variables.



You can also specify the:

- HOSToption, which displays only those system options in the host group.
- NOHOSToption, which displays only those system options in the portable group.

Environment variables are always returned.

LOGNUMBERFORMAT

Specifies that for options that return decimal numbers, the integer part and decimal part of the numbers are separated by a comma (,) rather than a period (.).

 **LOGNUMBERFORMAT** **Type:** Keyword

LONG

Specifies that the information returned includes the name of the system option, the current setting, and a description of the system option. This is the default level of detail returned.

 **LONG** **Type:** Keyword



The format of the information returned is one of the following ways, depending on the system option, and how it was set:

- *option-name description*
- *NOoption-name description*
- *option-name=value description*

option-name is the name of a system option, *value* is the value assigned to a system option and *description* is text that describes what the system option does.

NOEXPAND



Symbolic references are not expanded.

 **NOEXPAND** **Type:** Keyword

See `EXPAND` for more information on symbolic references.

NOHOST



Returns information only for portable system options.

 **NOHOST** **Type:** Keyword

System options are of two general types, host and portable. By specifying this option, only information about portable system options is returned. The information is provided in the same format as the `EXPAND` option.

NOLOGNUMBERFORMAT

Specifies that for options that return decimal numbers, the integer part and decimal part of the numbers are separated by a period (.) rather than a comma (,).

 **NOLOGNUMBERFORMAT** 

Type: Keyword

OPTION

Specifies the name of a system option for which you want information.

⇒ **OPTION** ⇒ = ⇒ *option-name* ⇒

Type: String

option-name is the name of a system option for which information is returned. You can specify more than one system option, but information is returned only for the last specified system option name.

If this option is used with `DEFINE`, `EXPAND`, `LONG`, `SHORT` or `VALUE`, information about *option-name* is returned in the same format as that option provides. By default, the option returns the same level of information as provided with `LONG`.

This option is ignored if used with `LISTGROUPS` and `GROUP`.

RESTRICT

Returns information about system options that are set as restricted. If a system option has been set as restricted, its value cannot be changed during the WPS session. System options can be restricted using the `SETINIT` file, configuration files, or the `OPTIONS` statement.

This option cannot be used with the `OPTION` option.

⇒ **RESTRICT** ⇒

Type: Keyword

If no system options have been restricted, the following message is returned:

```
The Site Administrator has not restricted any system options
```

For each system option that has been restricted, information is returned about the value of the option and how it was set. The same information as provided by `VALUE` is returned. For example, the following program restricts the system option `XWAIT`.

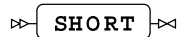
```
OPTIONS RESTRICT=XWAIT;  
PROC OPTIONS RESTRICT;
```

The `OPTIONS` procedure in the second line writes the following to the log:

```
Option value information for option XWAIT  
Option Value: XWAIT  
How option value set: Options statement
```

SHORT

Returns only a list of system options and their current values.



Type: Keyword

The system options are grouped by host and portable options, and environment variables.

The format of the information returned is one of the following ways, depending on the system option, and how it was set:

- *option-name*
- *NOoption-name*
- *option=value*

For example:

```
NOAUTOSIGNON  
BOMFILE  
BUFSIZE=0
```

If used with `OPTION`, only information for the specified system option is returned. For example:

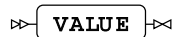
```
PROC OPTIONS OPTION=YEARCUTOFF SHORT;
```

returns the following:

```
YEARCUTOFF=1926
```

VALUE

Returns information on the value of system options and how they are set.



Type: Keyword

The information returned is in the format:

```
Option value information for option option-name  
Option Value: value  
How option value set: setting
```

where *option-name* is the name of a system option, and *value* is the value of the system option. System options can be set in various ways (as the default value, in a configuration file, from the command line, and so on). How the value is set is described by *setting*; this can be:

- Built-in default – the value is the default set when WPS was installed.
- SETINIT SITEOPTIONS statement – the value was set using SITEOPTIONS in a SETINIT file.
- Config file – the value was set in a configuration file.
- Restricted config file – the value was set in a restricted configuration file.
- Command line – the value was set through a command on the command line.
- Internal – the system option is internal to WPS and the value cannot be changed.

- `Options` statement – the value was set using the `OPTIONS` statement.
- `Optload` – the value was set using `PROC OPTLOAD`.

OPTLOAD procedure

Loads values for system options from a dataset.

You can use this procedure to set the system options for the current session. You might want to do this if you want to use saved system options, or if you want to run a program using the same system options on different servers.

You can create a dataset containing all system options and their corresponding values using the `OPTSAVE` [\(page 2407\)](#) procedure.

The dataset created by the `OPTSAVE` procedure contains two variables:

- `OPTNAME`, the name of a system option
- `OPTVALUE`, the value of the system option

Some system options might not be compatible with your version of WPS. Incompatible options are not loaded, and an error message is displayed in the log for each incompatible option. All other system options are, however, loaded. System options might be incompatible if they were specified using a later version of WPS, or using WPS on a different operating system.

Example – loading system options (page 2405)	2405
In this example, the procedure is used to load the values of system options.	
<code>OPTLOAD</code> procedure reference (page 2406)	2406
Loads values for system options from a dataset.	

Example – loading system options

In this example, the procedure is used to load the values of system options.

The values are retrieved from a dataset in which they were previously saved using the `OPTSAVE` procedure.

```
libname opts 'c:\temp';
proc optload data = opts.options;
```

The system options in the dataset `options` in the folder `c:\temp` are loaded for the current WPS session.

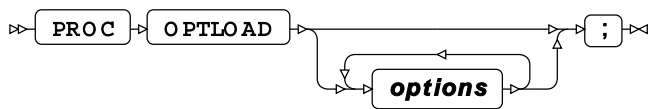
OPTLOAD procedure reference

Loads values for system options from a dataset.

PROC OPTLOAD [↗](#).....2406
 Invokes the OPTLOAD procedure.

PROC OPTLOAD

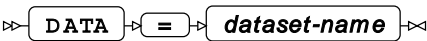
Invokes the OPTLOAD procedure.



Options

The following options are available with the PROC OPTLOAD statement

DATA



Type: String

The name of the dataset that contains the system options and their corresponding values.

The dataset name cannot be specified as an operating system pathname. It must be specified using a library name. The dataset can, therefore, be read from any file that can be accessed using a data engine. For example, the system options could be read from a Microsoft Excel spreadsheet or a database table.

If you specify more than one DATA option, only the final one is acted on.

An error occurs if:

- You do not specify this option
- You do not specify *dataset-name*
- The library reference or filename reference specifies a path that does not exist

OPTSAVE procedure

Saves the values of the current system options to a specified dataset.

You can use this procedure to save the system options for the current session. You might want to do this to save system options for later use, or to run a program using the same system options on different servers.

The dataset created by the OPTSAVE procedure contains two variables:

- OPTNAME, the name of a system option
- OPTVALUE, the value of the system option

You can load previously saved system options and their corresponding values using the OPTLOAD [\(page 2405\)](#) procedure.

Example – Saving system options in current session (page 2407)	2407
In this example, the procedure is used to save the current values of system options to the specified dataset.	
OPTSAVE procedure reference (page 2407)	2407
Describes the syntax and options for PROC OPTSAVE.	

Example – Saving system options in current session

In this example, the procedure is used to save the current values of system options to the specified dataset.

```
LIBNAME opts 'c:\temp';
PROC OPTSAVE OUT = opts.options;
```

The system options and values are saved in the dataset options in the folder c:\temp.

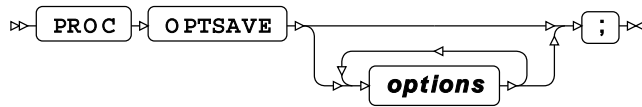
OPTSAVE procedure reference

Describes the syntax and options for PROC OPTSAVE.

PROC OPTSAVE (page 2408)	2408
Invokes the OPTSAVE procedure.	

PROC OPTSAVE

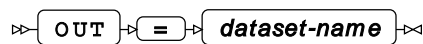
Invokes the OPTSAVE procedure.



Options

The following options are available with the `PROC OPTIONS` statement

OUT



Type: String

Specifies the name of the dataset to which system options and their values are written. This option is mandatory.

dataset-name cannot be specified as an operating system pathname, such as `c:\temp\optout.txt`. It must be specified using a library reference or filename reference. The dataset can, therefore, be written to any file that can be accessed using a data engine. For example, the system options could be written to a Microsoft Excel spreadsheet or database table.

If you specify more than one `OUT` option, only the final one is acted on.

An error occurs if:

- You do not specify this option
- You do not specify *dataset-name*
- The library reference or filename reference specifies a path that does not exist

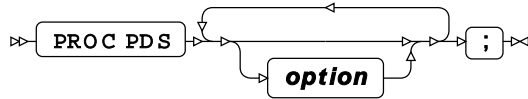
PDS procedure

Supported statements

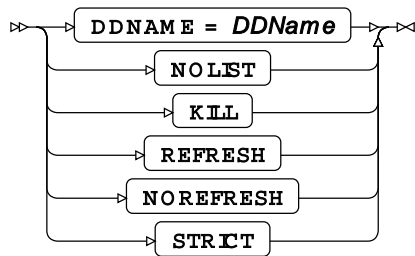
- `PROC PDS` [↗](#) (page 2409)
- `DELETE` [↗](#) (page 2409)
- `CHANGE` [↗](#) (page 2409)
- `EXCHANGE` [↗](#) (page 2409)

PROC PDS

Manages Partitioned Data Set Extended (PDS(E)) on Z/OS.

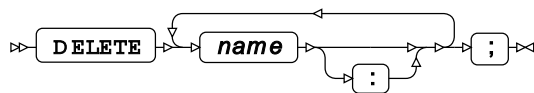


option



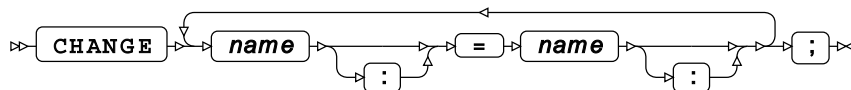
DELETE

Removes a member of a PDS(E).



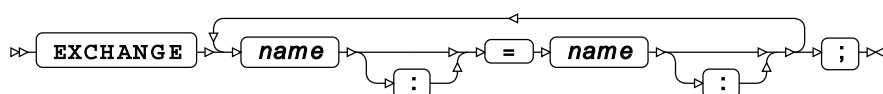
CHANGE

Changes a member of a PDS(E).



EXCHANGE

Exchanges names of pairs of a PDS(E) member.



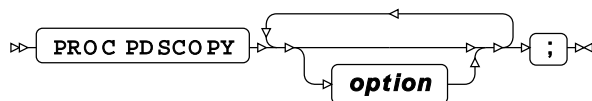
PDSCOPY procedure

Supported statements

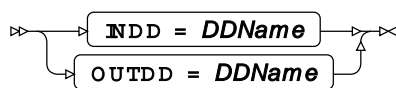
- *PROC PDSCOPY* [↗](#) (page 2410)
- *DELETE* [↗](#) (page 2410)
- *SELECT* [↗](#) (page 2410)
- *EXCLUDE* [↗](#) (page 2411)

PROC PDSCOPY

Copies a partitioned dataset.

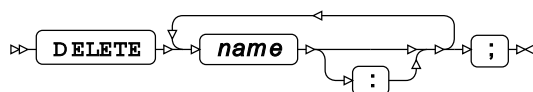


option



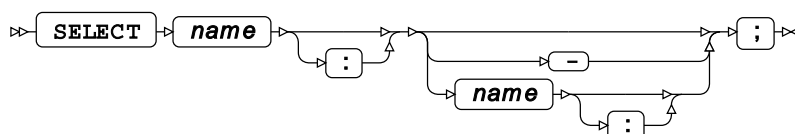
DELETE

Unsupported.



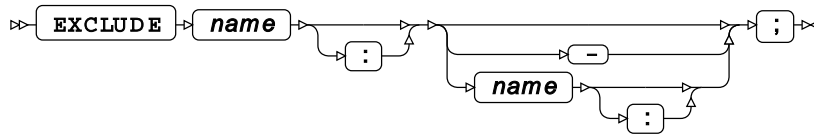
SELECT

Selects one or more names within a partitioned dataset, that may be copied. The colon (":") character is a wildcard.



EXCLUDE

Excludes one or more names within a partitioned dataset, from being copied. The colon (":") character is a wildcard.



PLOT procedure

Supported statements

- *PROC PLOT* [↗](#) (page 2411)
- *PLOT* [↗](#) (page 2412)
- *BY* [↗](#) (page 2412)
- *WHERE* [↗](#) (page 2413)

PROC PLOT

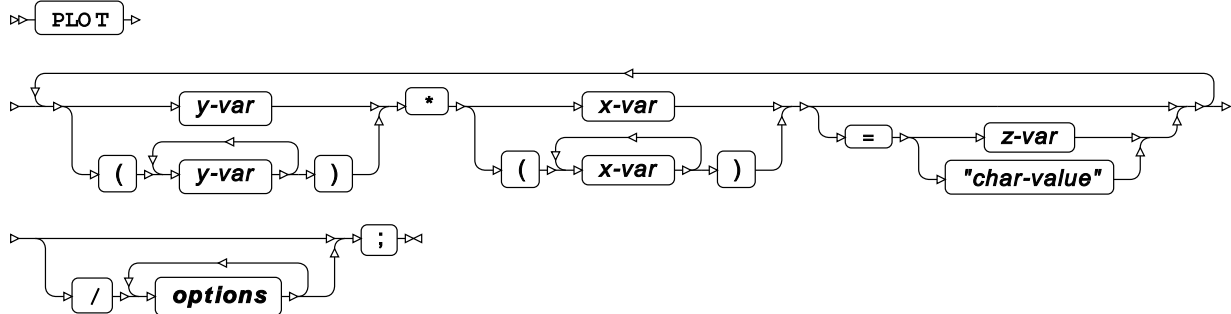
Generates textual plots from datasets.



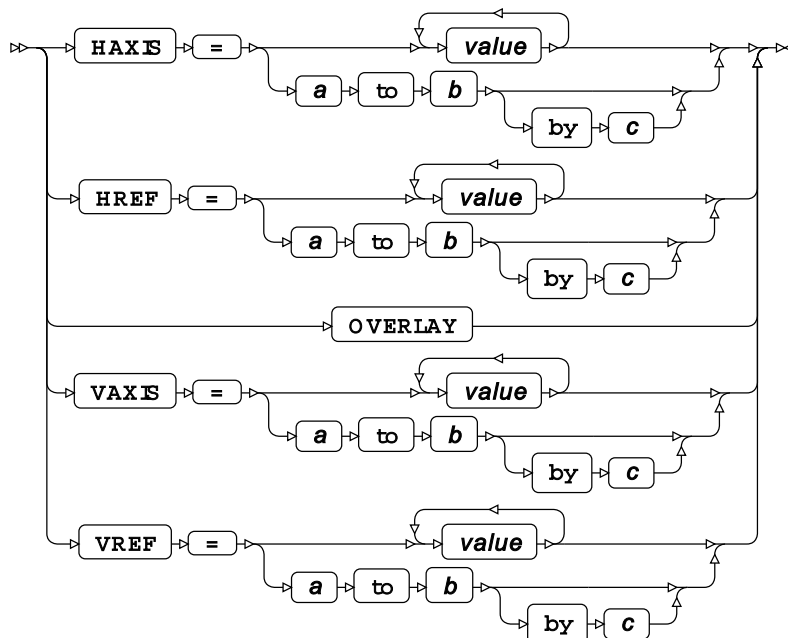
ⁱ See *Input dataset* [↗](#) (page 16).

PLOT

Defines the plots to be written to the log or a file.

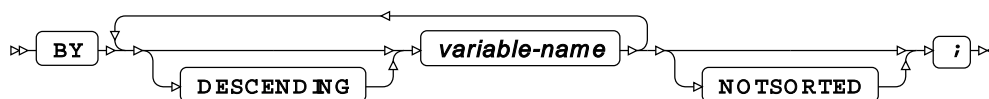


options



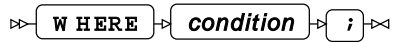
BY

Groups the observations in a dataset using one or more specified variables.



WHERE

Restricts the observations to be processed.



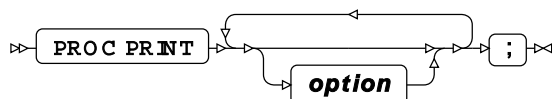
PRINT procedure

Supported statements

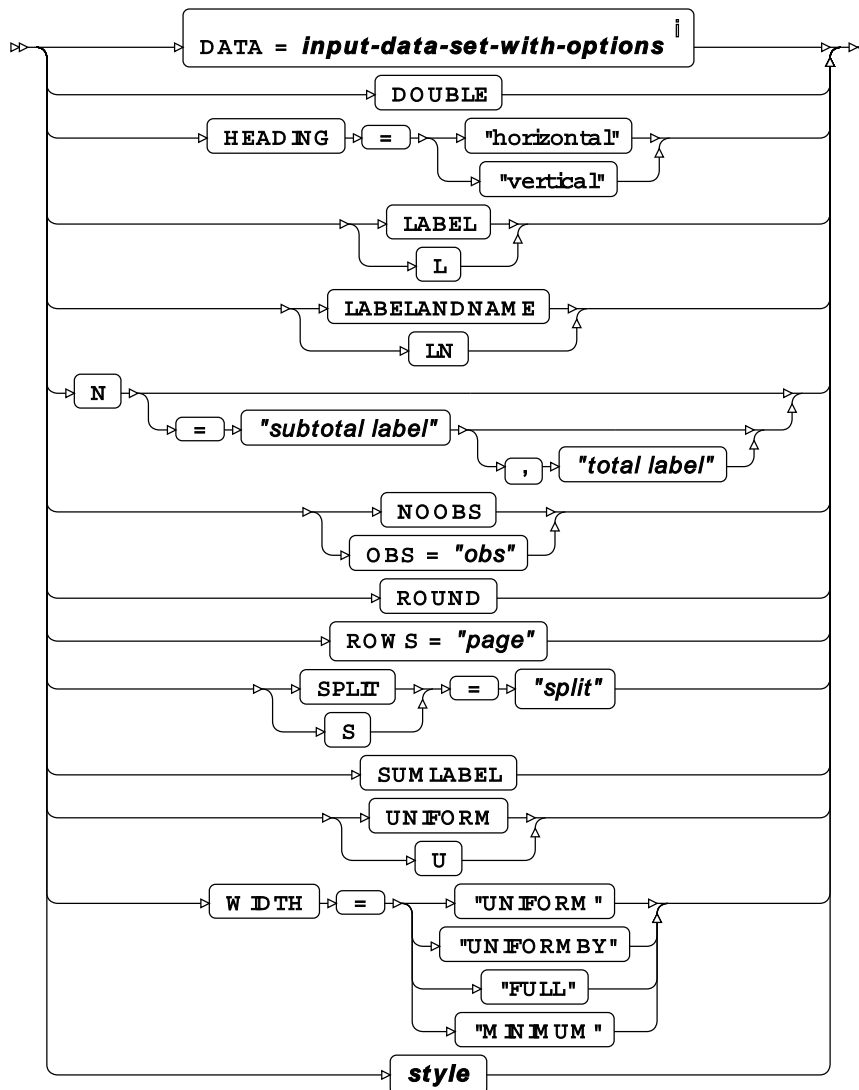
- *PROC PRINT* [↗](#) (page 2413)
- *ID* [↗](#) (page 2415)
- *PAGEBY* [↗](#) (page 2415)
- *SUM* [↗](#) (page 2416)
- *SUMBY* [↗](#) (page 2416)
- *VAR* [↗](#) (page 2416)
- *BY* [↗](#) (page 2416)
- *WHERE* [↗](#) (page 2416)

PROC PRINT

Prints observations from a given dataset to the ODS system.

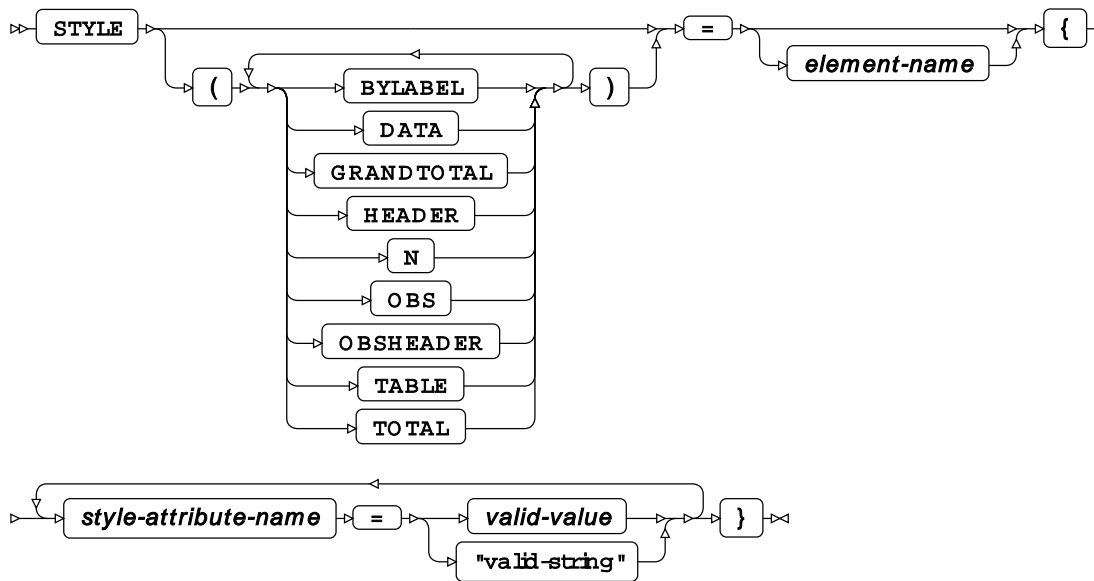


option



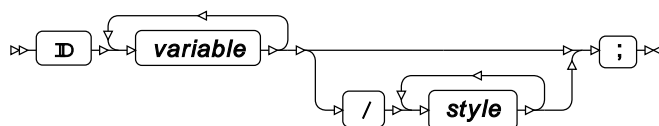
ⁱ See *Input dataset* [↗](#) (page 16).

style



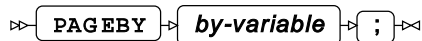
ID

Specifies one or more variables that identify the relevant observations in the output.



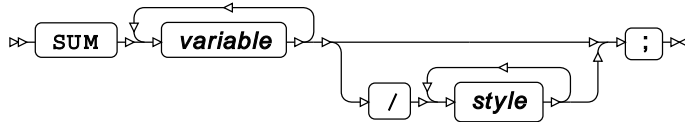
PAGEBY

Specifies a BY variable. When the BY variable changes (for example, A to B) a new page is created.



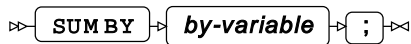
SUM

Sums one or more variables.



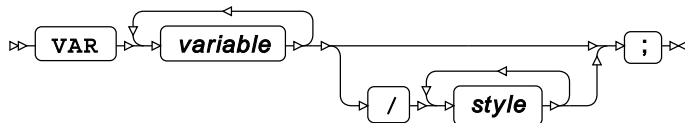
SUMBY

Sum all instances of the by-variable.



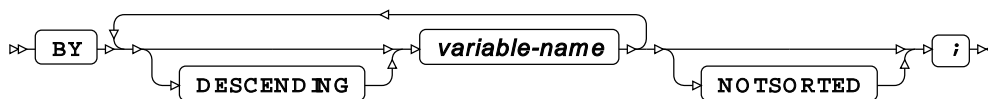
VAR

Specifies variables for which to calculate statistics to be printed.



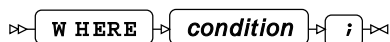
BY

Groups the observations in a dataset using one or more specified variables.



WHERE

Restricts the observations to be processed.



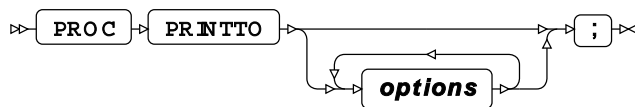
PRINTTO procedure

Supported statements

- *PROC PRINTTO* [↗](#) (page 2417)

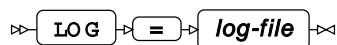
PROC PRINTTO

Redirects ODS listing or log output.



Options

LOG

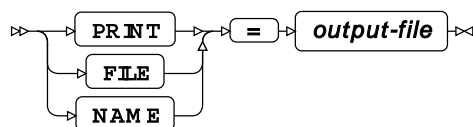


NEW



Type: Keyword

PRINT



UNIT



PWENCODE procedure

Encodes a password.

The password can be encoded using {sas001} (the default) or {sas003} encryption methods. The encoded password can be written to the log or to a file.

Example ↗	2418
In this example, the procedure is used to encode the specified password.	
PWENCODE procedure reference ↗	2418
Describes the syntax and options for PROC PWENCODE.	

Example

In this example, the procedure is used to encode the specified password.

The procedure is used twice, first to write the encoded password to the log, then to write it to a specified file.

```
PROC PWENCODE IN = "wombles" METHOD = SAS003;
PROC PWENCODE IN = "wombles" OUT='c:\temp\pw1.txt';
```

This produces the following output:

```
{sas003}d29tYmxlcw==
```

The second use of the procedure writes {sas001}d29tYmxlcw== to the file pw1.txt in the folder c:\temp.

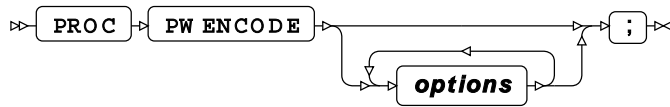
PWENCODE procedure reference

Describes the syntax and options for PROC PWENCODE.

PROC PWENCODE ↗	2419
Creates an encoded password and stores it in a file if required.	

PROC PWENCODE

Creates an encoded password and stores it in a file if required.



Options

The following options are available with the `PROC PWENCODE` statement

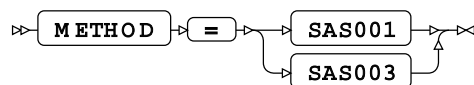
IN



Type: String

The password to be encoded. You do not have to enclose the password in quotation marks, unless the password contains special characters such as spaces, in which case you must enclose it in quotation marks. This option is mandatory.

METHOD



The method used to encode the password. This can be:

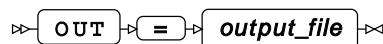
SAS001

Use the SAS001 encryption method. This is the default.

SAS003

Use the SAS003 encryption method.

OUT



Type: String

A file to which the encoded password is written. This is optional. If you do not specify a pathname, the file is written to the current working folder. If you are using Workbench, the working folder is the current project folder.

If you do not specify the `OUT` option, the encoded password is written to the log. If you do specify `OUT`, the filename overwrites any existing file with the same name. The procedure cannot append encrypted passwords to an existing file. Each file created by this procedure contains only one encrypted password.

The encoded password is prefixed with the encryption method, for example:

```
{sas001}dHdvIHdvbWJsZXM=
```

If you specify an option more than once, only the last instance of the option is used.

If you want to create the file in a location other than the current working or project folder, that folder must already exist.

PYTHON procedure

The PYTHON procedure enables WPS programs written in the SAS language to include code written in the Python language.

Combining the Python language and the SAS language enables the bulk of a data processing and analytics solution to be written in the industrial strength and high-performing SAS language, while also exploiting features present in the Python language.

Introduction ↗	2420
The Python procedure enables SAS language programs to include code written in the Python language.	
Setup and configuration ↗	2421
Setting the Python environment for WPS.	
Using Python with WPS ↗	2422
Using Python in a SAS language program enables you to make use of specialist Python packages such as <code>Scikit-learn</code> or <code>Tensorflow</code> .	
Example ↗	2427
Demonstrates how to use a SAS language dataset in <code>PROC PYTHON</code> to create a scatter plot diagram.	
Python procedure reference ↗	2429
Describes the syntax and options for <code>PROC PYTHON</code> and its contained statements.	

Introduction

The Python procedure enables SAS language programs to include code written in the Python language.

By combining Python and the SAS language you can:

- Use the SAS language to bulk process and prepare data, and pass the processed data to Python.
- Use Python packages you have previously developed for data analysis.
- Use Python data analysis packages or solutions that may not be available in the SAS language.

Data is passed between the SAS language and Python language environments using the `EXPORT` statement. Once data has been transferred, that data is made available as a pandas DataFrame to a Python program. On completion of the Python program data can, if required, be returned to the SAS language environment using the `IMPORT` statement.

Setup and configuration

Setting the Python environment for WPS.

When using Python with WPS:

- The 32-bit Python interpreter is required for the 32-bit version of WPS, and the 64-bit Python interpreter is required for the 64-bit version of WPS.
- The `pandas` package must be installed with the Python interpreter. This can be checked using the `pip` utility by running `pip list` on the command line.

The WPS distribution does not include either the Python interpreter or `pandas` package. If you do not have Python installed, you can obtain a copy of the Python interpreter from <https://www.python.org> or install a packaged Python environment that includes the necessary modules.

The `PYTHON` procedure can be used with Python version 3.5.0 and later, and is currently supported on Microsoft Windows, Linux-based systems, and macOS.

The procedure is not currently supported on IBM mainframe.

Python environment variables

You must set the `PYTHONHOME` environment variable for WPS to locate and use the Python interpreter. This variable must reference the folder where the main Python library is located – for example, `python3.dll` on Microsoft Windows.

Standard output and error streams

The Python standard output stream (`sys.stdout`) and standard error stream (`sys.stderr`) are redirected to WPS output when the procedure is run:

- `sys.stderr` is redirected to the WPS log file.
- `sys.stdout` is redirected to the WPS listing file when WPS is run on the command line, or to all specified ODS output destinations when Workbench is used.

If you use the Python `print()` function to put output to the WPS listing file, the number of characters appearing in the listing output is determined by the WPS `LINESIZE` system option. The `LINESIZE` option can be used to print strings of up to 256 characters; the use of the `print()` function should therefore be for limited information such as log statements.

If you want to return large volumes of Python-generated content to WPS, create a DataFrame containing the required content and import the DataFrame into WPS using the `IMPORT` statement.

The following example creates a list of functions available in the Python pandas package. The list is printed using `print (fnList)` and also converted to a DataFrame and imported into WPS.

```
PROC PYTHON;
SUBMIT;
import inspect
fnPandas = inspect.getmembers(pandas, inspect.isfunction)
fnList = [fn[0] for fn in fnPandas]
print (fnList)
fnTable = pandas.DataFrame({'function': fnList})
ENDSUBMIT;
IMPORT DATA=PANDAFN PYTHON=fnTable;
RUN;
```

The printed output truncates the list of functions:

```
['Expr', 'Term', 'bdate_range', 'concat', 'crosstab',
 'cut', 'date_range', 'eval', 'factorize',
```

When converted to a DataFrame and imported as a WPS dataset, all available functions are listed.

Using Python with WPS

Using Python in a SAS language program enables you to make use of specialist Python packages such as `Scikit-learn` or `Tensorflow`.

The first time the Python procedure is invoked in a SAS language program, WPS automatically imports the `pandas` and `numpy` packages. You can access the functionality in the `pandas` and `numpy` packages in an in-line Python language program – written between the `SUBMIT` and `ENDSUBMIT` statements – by referencing the fully-qualified package, class or function name, for example:

```
PROC PYTHON;
SUBMIT;
content = pandas.read_csv('file:///C:/project/sourcedata/wps.csv')
...
ENDSUBMIT;
RUN;
```

Alternatively, you can use the `import ... as ...` statement to alias either the `pandas` or `numpy` package name, for example:

```
PROC PYTHON;
SUBMIT;
import pandas as pd
content = pd.read_csv('file:///C:/project/sourcedata/wps.csv')
...
ENDSUBMIT;
RUN;
```

Other Python packages can be imported and used within the in-line Python code, for example:

```
PROC PYTHON;  
  EXPORT DATA=source;  
  SUBMIT;  
import statsmodels.formula.api as lm  
result = lm.ols(formula='x ~ y + z', data=source).fit()  
...  
  ENDSUBMIT;  
RUN;
```

Each subsequent use of the PYTHON procedure in a SAS language program can use the same Python environment. This means any global variables or imported packages used in a PYTHON procedure invocation are available to all subsequent PYTHON procedure invocations.

Each PYTHON procedure invocation can include multiple blocks of in-line Python language code, and use a combination of in-line Python language code, and Python programs run using the `EXECUTE` statement.

Data type conversion

Describes the correspondence between SAS language formats and pandas data types.

This section describes the correspondence between WPS formats and Python pandas data types. WPS has many formats that affect the output and display of data. When you write data to a pandas DataFrame using the Python procedure `EXPORT` statement, formatted data is converted to an equivalent pandas or numpy data type.

Many formats only affect the layout of data output, such as adding currency symbols or comma separators, and these formats have no effect when writing data.

SAS language unformatted data to Python

Unformatted data is converted to a pandas DataFrame type as follows:

WPS format	Python data type	Notes
Unformatted number	float64	
Unformatted string	object	The maximum object length for a variable is not known as part of the dataset metadata.

SAS language formatted data to Python – numbers

The core numeric formats are converted to a pandas DataFrame type as follows:

WPS format	Python data type	Notes
<i>w.d</i>	Float64	
BEST. and BEST <i>w.</i>	float64	
FLOAT <i>w.d</i>	float64	

SAS language formatted data to Python – strings

The core character formats are converted to a pandas DataFrame type as follows:

WPS format	Python data type	Notes
\$ <i>w.</i> \$CHAR <i>w.</i> \$F <i>w.</i>	object	The maximum object length for a variable is not known as part of the dataset metadata.

SAS language formatted data to Python – dates and times

Date, datetime and time formats are converted to a pandas DataFrame type as follows:

WPS format	Python data type	Notes
DATE <i>w.</i>	datetime64[ns]	Numpy datetime type.
DDMMYY <i>w.</i> and all variants (such as DDMMYYB <i>w.</i> , MMDDYY <i>w.</i> , and YYMM <i>w.</i>)	datetime64[ns]	Numpy datetime type.
DTDATE <i>w.</i> and all variants (such as DTMONYY <i>w.</i> and DTWKDATX <i>w.</i>	datetime64[ns]	Numpy datetime type.
TIME <i>w.</i> , HOUR <i>w.</i> , HHMM <i>w.</i> and all similar time formats.	float64	
JULIAN <i>w.</i> and all similar date formats.	datetime64[ns]	Numpy datetime type.

Python pandas data types to SAS language dataset

SAS language datasets only contain numeric and character data. Formats might be applied to the data in the dataset to more closely represent the source data from Python. Importing a pandas DataFrame using the `IMPORT` statement converts data as follows:

Pandas data type	WPS format	Notes
Object	Character	The maximum object length is calculated before importing.
int64	Numeric	
bool	Numeric	<i>True</i> is converted to 1; <i>False</i> is converted to zero (0).

Pandas data type	WPS format	Notes
Float64	Numeric	
datetime64[ns]	Numeric	Formatted as DATETIME19.

Some values in Python cannot be represented in the same manner as a SAS language datasets. The following table shows how these values are converted:

Pandas data value	WPS value	Notes
null string (' ')	' '	Missing character value.
True	1	Numeric value
False	0 (zero)	Numeric value
NaN	.	Missing numeric value
Defined with <code>float('nan')</code>	.	
Infinity	.	Missing numeric value
Defined with <code>float('inf')</code>	.	
Defined with <code>float('-inf')</code>	.	

Import custom Python modules

How to use your own packages and modules in Python.

To use custom packages, they must be accessible to Python. The paths to locations containing custom packages are specified by the Python `sys.path` variable. The `sys.path` variable constructs a list of locations to search using the `PYTHONHOME` and `PYTHONPATH` environment variables, or the locations can be specified by modifying the value of the variable during the execution of a program.

- `sys.path` contains a list of folders searched for packages. The variable is constructed from the values in `PYTHONHOME` and `PYTHONPATH`.
- `PYTHONHOME` specifies the location of the Python interpreter and standard libraries, including packages installed into *site-packages* using a package manager such as *PIP*. The variable must be specified for WPS to interact with Python.
- `PYTHONPATH` specifies a list of folders to search for packages. This list is prepended to the search list defined in `sys.path`.

The first item in `sys.path` is either the directory containing the python program or an empty string, interpreted by Python as the current directory. When run from WPS, `sys.path[0]` contains the first search path specified in either `PYTHONPATH` or `PYTHONHOME`. If your program references packages in the current directory, you must modify `sys.path` when `PROC PYTHON` is running. See [Modifying the `sys.path` variable](#) (page 2426)

Set `PYTHONPATH` before running WPS

The `PYTHONPATH` variable can be defined as a system variable, and used by all applications on your device that run Python.

If you have multiple installations of Python, for example Python 2 and Python 3, setting *PYTHONPATH* using a system variable might cause your program to attempt to import the incorrect version of a package. In these circumstances, you should set the variable as part of the SAS language program you run in WPS.

Set *PYTHONPATH* in a SAS language program

The *PYTHONPATH* can be set in a SAS language program using the *SET* system option. For example, to use packages stored in *C:\temp\python* folder, the following can be added to a program before the *PROC PYTHON* statement:

```
OPTIONS SET = PYTHONPATH 'C:\temp\python';
```

If *PYTHONHOME* is specified as *C:\python3*, the content of the Python *sys.path* variable is:

```
['C:\\temp\\python', 'C:\\python3\\python37.zip',  
'C:\\python3\\DLLs', 'C:\\python3\\lib', 'C:\\python3',  
'C:\\python3\\lib\\site-packages']
```

Modifying the *sys.path* variable

The Python *sys.path* variable can be modified programmatically by adding the following to a Python language program:

```
import os, sys  
sys.path.append(os.getcwd())
```

If a Python language program run using the *EXECUTE* statement includes other Python files, add the above as an in-line program before the executed program to enable WPS to locate the included files:

```
PROC PYTHON;  
  SUBMIT;  
import os, sys  
sys.path.append(os.getcwd())  
  ENDSUBMIT;  
  EXECUTE 'programs/dataCollect.py';  
RUN;
```


Using graphics created by Python

Any graphics generated using functionality in Python are captured and can be included in the WPS session's standard ODS output.

WPS creates a temporary variable, *wpsgloc* pointing to a temporary folder that is used to store graphics for inclusion in ODS output. The variable must be added to the name of image file every time you create an image. This can be done using, for example, the `os.path.join()` Python function:

```
PROC PYTHON;
SUBMIT;
import os
import matplotlib.pyplot as plt
...
plt.savefig(os.path.join(wpsgloc, 'my_image.png'))
ENDSUBMIT;
RUN;
```

The *wpsgloc* variable cannot be modified in the SAS language program, and is either:

- The WORK location when run in Workbench, for example:

```
C:\Users\user-id\AppData\Local\Temp\WPS Temporary Data
```

- The working directory when WPS Analytics in run on the command line.

```
ODS PDF FILE='scatter_plot.pdf';
PROC PYTHON;
  EXPORT DATA=STATS PYTHON=df;
  SUBMIT;
import os
import matplotlib.pyplot as plt
df.plot(kind='scatter', x='i', y='j')
plt.savefig(os.path.join(wpsgloc, 'scatter.png'))
ENDSUBMIT;
RUN;
ODS PDF CLOSE;
```

wpsgloc is a temporary location for graphics images used in Workbench.

When run on the command line, the graphics are created in the current directory. The graphics are referenced in HTML output and copied into PDF output.

Example

Demonstrates how to use a SAS language dataset in `PROC PYTHON` to create a scatter plot diagram.

The following example creates a dataset in a SAS language `DATA` step, and then uses the `EXPORT` statement to pass that dataset to the Python environment. The dataset is converted to a pandas DataFrame as part of the export, and the DataFrame is used to create a scatter plot using Matplotlib.

An output PDF file destination is created using the SAS language Output Delivery System (ODS). Adding PDF to the output destinations includes the returned scatter plot image file in the PDF output. The PDF is saved and the output can be viewed in a PDF viewer.

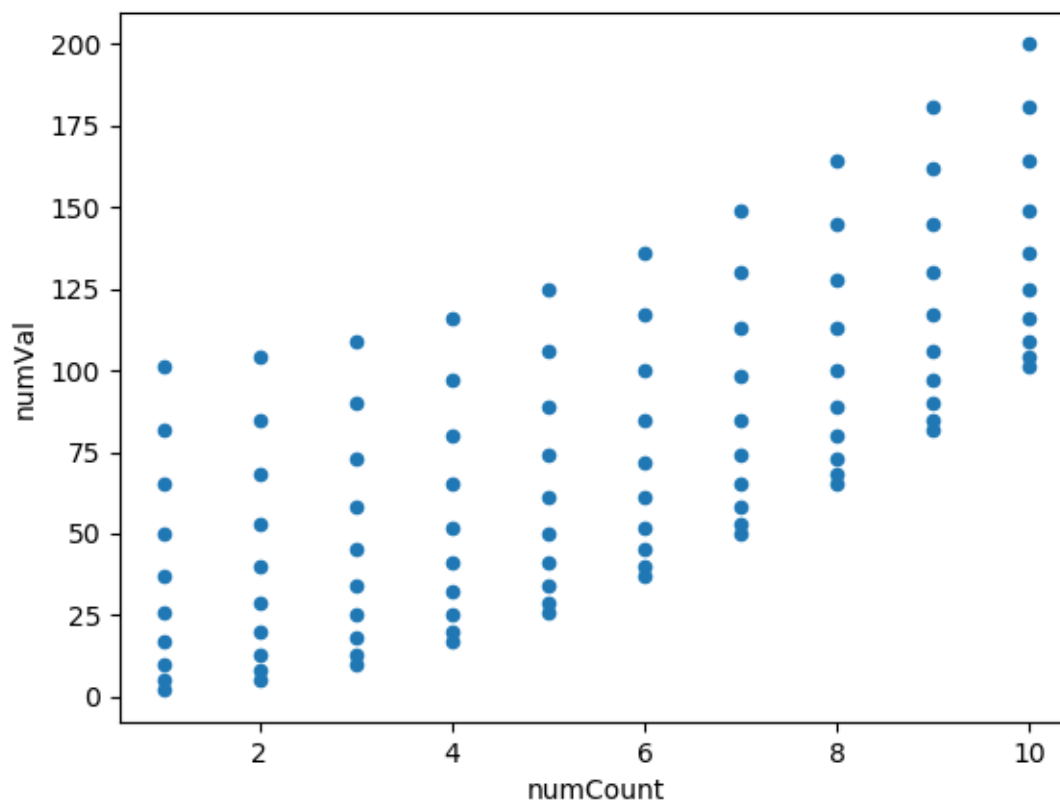
This example requires the following Python packages:

- SciPy
- Matplotlib

```
ODS PDF FILE='scatter_plot.pdf';
DATA stats (drop = count);
  DO count=1 TO 10;
    DO numCount=1 TO 10;
      numVal = (numCount*numCount)+(count*count);
      OUTPUT;
    END;
  END;
RUN;

PROC PYTHON;
  EXPORT DATA=STATS PYTHON=df;
  SUBMIT;
import os
import matplotlib.pyplot as plt
df.plot(kind='scatter', x='numCount', y='numVal')
plt.savefig(os.path.join(wpsgloc, 'scatter.png'))
ENDSUBMIT;
RUN;
ODS PDF CLOSE;
```

This creates the following scatter plot in the ODS PDF output:



Python procedure reference

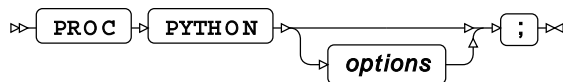
Describes the syntax and options for `PROC PYTHON` and its contained statements.

<code>PROC PYTHON</code> ↗	2430
Invokes the Python environment that enables the execution of in-line or external Python language programs.	
<code>EXECUTE</code> ↗	2431
Runs a Python program stored in a separate file.	
<code>EXPORT</code> ↗	2433
Converts a SAS language dataset to a pandas <code>DataFrame</code> .	
<code>IMPORT</code> ↗	2434
Enables a pandas <code>DataFrame</code> to be converted to a SAS language dataset and referenced in a SAS language program.	
<code>SUBMIT</code> ↗	2435
Specifies the start of an in-line Python language program.	

ENDSUBMIT [↗](#)..... 2435
Specifies the end of an in-line Python language program.

PROC PYTHON

Invokes the Python environment that enables the execution of in-line or external Python language programs.



Datasets created in WPS can be made available to the Python program using the `EXPORT` statement, and a dataset imported from the Python program into WPS using the `IMPORT` statement.

A Python program can be either written in-line in the `PYTHON` procedure, or run from a separate file:

- To run an in-line Python program, use the `SUBMIT` and `ENDSUBMIT` statements.
- To run a Python program stored in an external file use the `EXECUTE` statement.

The Python environment is exited using a `RUN` statement.

When the Python environment is invoked, the `pandas` and `numpy` packages are automatically loaded. These packages can be used either by quoting the full package name in code, or by using the `import ... as ...` statement to alias the package name.

Options

The following options are available with the `PROC R` statement.

KEEP

Specifies that the current Python environment is not terminated when the procedure exits.



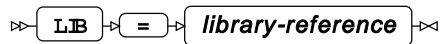
When specified, the current Python environment is kept active when the current procedure exits, and the environment is used in the next invocation of the Python procedure in the same program. If that invocation does not specify `KEEP`, the environment is terminated when the procedure exits.

The default behaviour is to terminate the Python environment at the end of the procedure. Specifying `KEEP` keeps the current Python environment, including any modules loaded during the execution of a Python program, to be used in the next invocation of `PROC PYTHON`.

You can specify the `PYTHONKEEP` system option to use the same Python environment for the duration of the execution of the SAS language program.

LIB

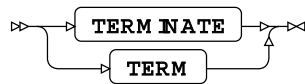
Specifies the default library location for the procedure step. The default location is the `WORK` library.



The **LIB** location is used when *libname* is not specified as part of the path for the **DATA** option of either the **EXPORT** or **IMPORT** statements.

TERMINATE

Specifies that the Python environment is terminated when the procedure exits.



Specifying **TERMINATE** stops the current Python environment even if the **PYTHONKEEP** system option has been specified. A subsequent invocation of **PROC PYTHON** in the same program only has the default **pandas** and **numpy** packages loaded.

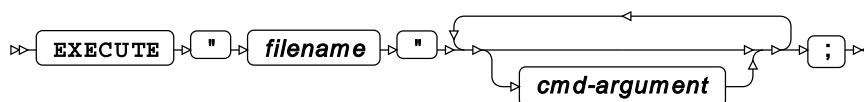
Example

The following example shows how to invoke the **PYTHON** procedure to print `hello world` to ODS output.

```
PROC PYTHON;  
SUBMIT;  
print ('Hello World')  
ENDSUBMIT;  
RUN;
```

EXECUTE

Runs a Python program stored in a separate file.



The **EXECUTE** statement is an alternative to using the **SUBMIT** statement. It enables Python code placed in a separate file to be run WPS Analytics.

Execute options

The following options are available with the **EXECUTE** statement.

filename

A string, in quotation marks, containing the path of the Python program file. *filename* can be either an absolute pathname or a relative pathname.

In Workbench, the path for relative file pathnames is the Workspace. For example, to run a file named `myProgram.py` from a project named `python`, the relative path is `/python/myProgram.py`.

cmd-argument

Specifies a command line argument passed to the Python program. Arguments are accessed in the programing from `sys.argv`.

All command line arguments are passed to the Python program as strings. Numeric arguments must be in quotation marks, and the Python program must convert the arguments to the required numeric type.

Basic example

In this example, a Python program stored in an external file is executed in the `PYTHON` procedure. The file is referenced using the absolute path:

```
PROC PYTHON;  
    EXECUTE 'C:\temp\printDS.py';  
RUN;
```

Example – pass an argument to a Python program

In this example, an external Python program `multiple.py` is executed that returns the square of a specified value. The value is specified in a variable.

```
import sys  
  
def multiply (value):  
    return value*value  
  
print(multiply(int(sys.argv[0])))
```

`multiple.py` is executed from the `PYTHON` procedure in a SAS language program, and the required value to multiply is passed to the Python program as an argument to the `EXECUTE` statement. The numeric argument `6` is passed as a string. The Python program uses the built-in `int()` function to return an integer before calculating the square.

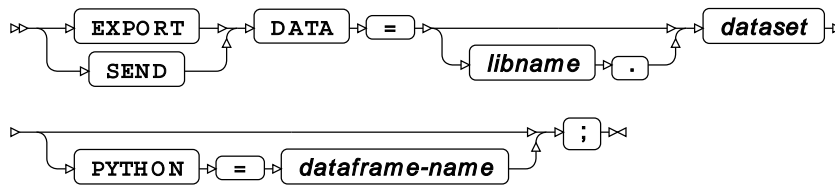
```
PROC PYTHON;  
    EXECUTE 'C:/temp/multiple.py' '6';  
RUN;
```

This produces the following in ODS output:

```
36
```

EXPORT

Converts a SAS language dataset to a pandas DataFrame.



Dataset preparation should be completed before exporting the data to Python. This enables you to use the dataset processing capability of the SAS language to create an export dataset containing only the required data for the Python language program.

Export options

The following options are available with the `EXPORT` statement.

DATA

Specifies the dataset location and name of the SAS language dataset to be converted.

The library location can be specified using either *libname* in the `DATA` option, or the `LIB` option of the `PROC PYTHON` statement.

- If *libname* is specified, that location is always used.
- If the `LIB` option of the `PROC PYTHON` statement is specified and *libname* is not specified, the location in the `LIB` option is used.
- If neither *libname* nor the `LIB` option on the `PROC PYTHON` statement are specified, the `WORK` location is used.

PYTHON

Specifies the name of the pandas DataFrame as used in the Python language program.

The pandas DataFrame name in Python language code is case sensitive, and the *dataframe-name* specified must match the case used of the Python variable name.

If this option is not specified, the *dataframe-name* default is the *dataset* name specified in the `DATA` option. If you use the default *dataset* name in an in-line Python language program, the variable name must match the case used in the `DATA` option.

Example – export a SAS language dataset to a pandas DataFrame

The following example creates a dataset in a SAS language `DATA` step. The dataset is then exported to a pandas `DataFrame` and the column types printed out.

```
DATA TESTDATA;
INFILE CARDS DLM='#';
INPUT num char $;
CARDS;
1 # Hello
2 # World
;
RUN;

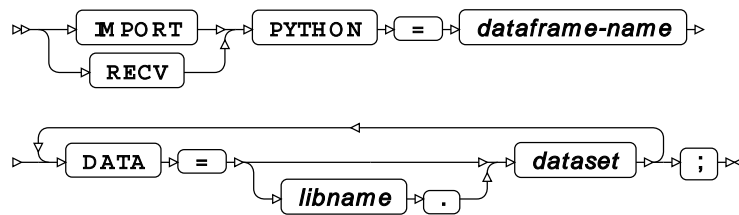
PROC PYTHON;
EXPORT DATA=TESTDATA PYTHON=dframe;
SUBMIT;
print (dframe)
ENDSUBMIT;
RUN;
```

Which writes the following to ODS output:

	num	char
0	1.0	Hello
1	2.0	World

IMPORT

Enables a pandas `DataFrame` to be converted to a SAS language dataset and referenced in a SAS language program.



Import options

The following options are available with the `IMPORT` statement.

DATA

Specifies the dataset location and name as used in the WPS Analytics SAS language environment.

The library location can be specified using either *libname* in the `DATA` option, or the `LIB` option of the `PROC PYTHON` statement.

- If *libname* is specified, that location is always used.

- If the `LIB` option of the `PROC PYTHON` statement is specified and *libname* is not specified, the location in the `LIB` option is used.
- If neither *libname* nor the `LIB` option on the `PROC PYTHON` statement are specified, the `WORK` location is used.

PYTHON

Specifies the name of the pandas `DataFrame` as used in the Python environment. Must be specified.

dataframe-name is case sensitive and must match the case used for the imported pandas `DataFrame` in the Python program.

SUBMIT

Specifies the start of an in-line Python language program.

» SUBMIT ¶ ; «

An in-line program is defined as part of the `PYTHON` procedure in a SAS language program. The `SUBMIT` statement marks the start of the program, and the `ENDSUBMIT` statement marks the end.

The Python language program must start on a new line after the `SUBMIT` statement, and the `ENDSUBMIT` statement must appear at the beginning of a line on its own. The first line of the Python program code must not start with any white space and any subsequent statements must follow the normal Python requirements for indentation, for example:

```
PROC PYTHON;
  SUBMIT;
  fruits = ['apple', 'banana', 'cherry', 'damson', 'elderberry', 'fig']
  for fruit in fruits:
    print(fruit)
  ENDSUBMIT;
RUN;
```

Multiple in-line Python language programs can exist in a single `PYTHON` procedure. Each Python language program is executed as it is encountered. Variables defined in one in-line language program can be used in subsequent in-line programs in the same Python procedure.

ENDSUBMIT

Specifies the end of an in-line Python language program.

» ENDSUBMIT ¶ ; «

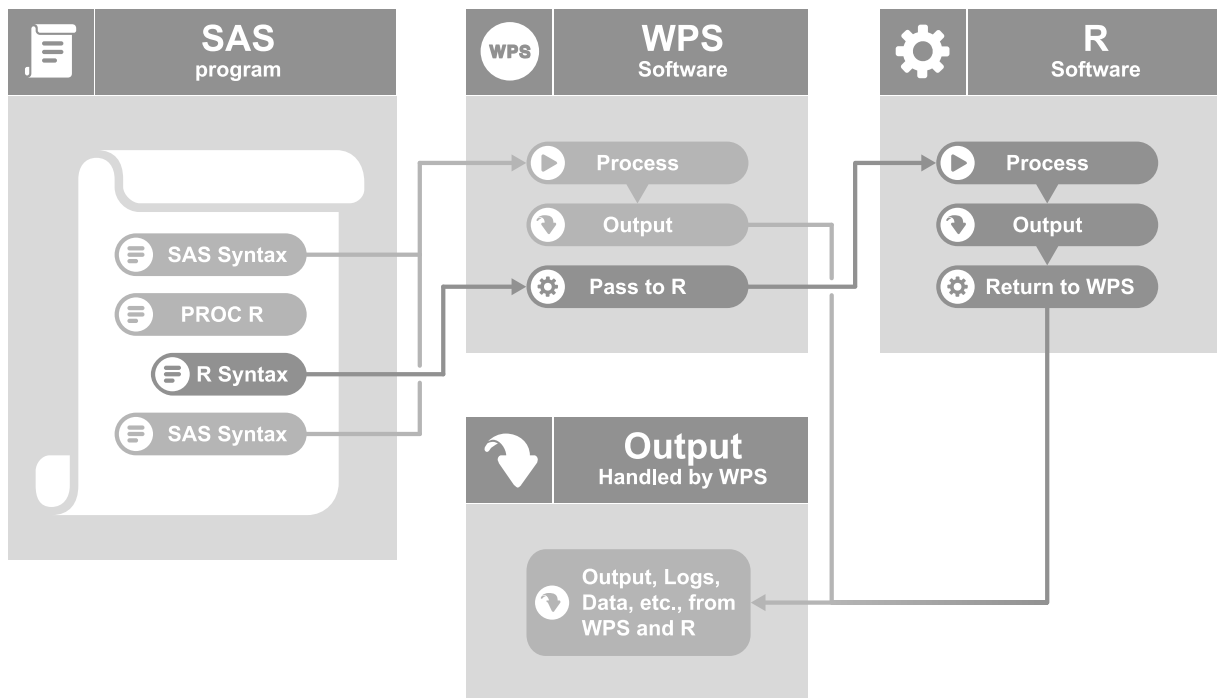
The `ENDSUBMIT` statement must be entered at the start of a new line after the Python language program.

R Procedure

The R procedure enables WPS programs written in the SAS language to include code written in the R language.

Combing the R language and the SAS language enables the bulk of a data processing and analytics solution to be written in the industrial strength and high-performing SAS language, while also exploiting features present in the R language.

The following image shows how a SAS language program using the R procedure is processed.



Introduction ↗	2437
The R procedure enables SAS language programs to include code written in the R language.	
Setup and configuration ↗	2437
Setting the R environment for WPS.	
Using R with WPS ↗	2439
Using R in a SAS language program enables you to use features that might not be available in WPS.	
Example ↗	2443
Demonstrates how to use a SAS language dataset in the R procedure to create a scatter plot diagram.	

R procedure reference ↗	2444
Describes the syntax and options for PROC R and its contained statements.	

Introduction

The R procedure enables SAS language programs to include code written in the R language.

By combining R and the SAS language you can:

- Use the SAS language to bulk process and prepare data, and pass the processed data to R.
- Use R packages you have previously developed for data analysis.
- Use R data analysis packages or solutions that may not be available in the SAS language.

We recommend that programs are mainly coded in the language of SAS, using R where specialist statistics are required.

Data is passed between the SAS language and R language environments using the `EXPORT` statement. Once data has been transferred, that data is made available as a `data.frame` object to an R program. On completion of the R program data can, if required, be returned to the SAS language environment using the `IMPORT` statement.

Setup and configuration

Setting the R environment for WPS.

When using R with WPS:

- The 32-bit R interpreter is required for the 32-bit version of WPS, and the 64-bit R interpreter is required for the 64-bit version of WPS.
- You do not have to install extra modules, and there are no special licensing requirements to use the R procedure.

The R procedure can be used with R version 2.15.x and later, and is currently supported on Microsoft Windows, Linux-based systems, and macOS.

WPS software is not shipped with a copy of R. To use the R procedure, you need a separate installation of R.

Following R installation, set the `R_HOME` environment variable to point to the folder containing either `libr.dll` on Windows platforms or `libr.so` on UNIX or Linux platforms.

Installing the R interpreter

Windows platforms

The Windows installer package can be downloaded from <http://www.r-project.org/> and includes both 32-bit and 64-bit versions of R. The same installer can be used for both 32-bit and 64-bit versions of the WPS software.

By default, the R installation saves the installation location in the Windows registry which is where WPS looks to identify the currently installed version. This is the version of R most recently installed, and no special configuration of WPS is required for WPS to locate it.

UNIX or Linux platforms

WPS requires the shared R library `libR.so` to interact with the R interpreter. This library is not included with the R binary distribution for UNIX platforms by default. On UNIX or Linux platforms, you need to either build R from source to include the required shared library, or install R using your systems package management system.

To build from source code:

1. You require a minimal set of pre-installed libraries before you can build R from source code. These are equivalent to the *build-essential* package plus a JDK on Ubuntu.
2. Download the R source code from <http://www.r-project.org/> and follow the instructions included with the download. Ensure you use the `--re-establishment` option when running `configure` to build the `libR.so` shared library, for example:

```
./configure --enable-R-shlib --prefix=$HOME/R
```

For more information, see the R documentation at <http://www.r-project.org/>.

macOS platforms

The binary distribution of R can be installed directly from the R project website.

To use R with WPS, the `R_HOME` variable must be set to point to the installation directory containing the `libR.so` shared library. The default setting is: `/Library/Frameworks/Framework/Resources`. Because of the way applications are launched on macOS, it is not possible to set `R_HOME` in a shell profile script.

To use a specific version of R, you can modify the setting appropriately, for example: `/Library/Frameworks/Framework/Versions/3.0/Resources`

Setting the *R_HOME* environment variable

To locate the installed version of R, the *R_HOME* environment variable must be set.

On Windows platforms, the *R_HOME* environment variable must point to the folder containing the `libr.dll` file. On UNIX or Linux platforms, the *R_HOME* environment variable must point to the folder containing the `libr.so` file.

If you are running WPS with R on a Unix or Linux platform, set the *R_HOME* variable to point to the folder containing `libr.so`.

If you have multiple installations of R, set *R_HOME* variable as part of the SAS language program you run in WPS.

Set *R_HOME* before running WPS

The *R_HOME* variable can be defined as a system variable, and used by all applications on your device that run R. It is not necessary to set the variable on the Windows platform if the default R installation location is used.

Set *R_HOME* in a SAS language program

The *R_HOME* variable can be set in a SAS language program using the `SET` system option, for example:

```
OPTIONS SET = R_HOME 'C:\Program Files\R\R-3.5.0';
```

This will set the *R_HOME* environment variable for duration of the execution of the SAS language program.

Using R with WPS

Using R in a SAS language program enables you to use features that might not be available in WPS.

Installed R packages can be imported and used within the in-line R code using the library statement, for example:

```
PROC R;  
  SUBMIT;  
    library(datasets)  
    data(iris)  
    summary(iris)  
    plot(iris)  
  ENDSUBMIT;  
RUN;
```

Each subsequent use of the R procedure in a SAS language program can use the same R environment. This means any global variables or imported packages used in an R procedure invocation are available to all subsequent R procedure invocations.

Each R procedure invocation can include multiple blocks of in-line R language code, and use a combination of in-line R language code, and R programs run using the `EXECUTE` statement.

Data type conversion

Describes the correspondence between SAS language formats and R data types.

When you write data to a `data.frame` using the R procedure `EXPORT` statement, formatted data is converted to an equivalent R data type. WPS has many formats that affect the output and display of data. Formats that only affect the layout of data output, such as adding currency symbols or comma separators, have no effect when writing data.

A `data.frame` is imported using the R procedure `IMPORT` statement. Any object that can be coerced into a `data.frame` using the `as.data.frame` R function is imported into the WPS dataset.

Logical values

Logical values are converted into numeric values in the WPS dataset. The values of vectors of type logical are converted as follows:

R Value	WPS Value
TRUE	1
FALSE	0

Integer values

The R language value `NA`, which is represented as the minimum integer value (-2147483648) is converted to a SAS language missing value.

Real values

There are three special real numeric values in the R language:

- `NA` (Not Available). Represents an absent value.
- `NaN` (Not a Number). Represents an undefined value, or a value that cannot be displayed, for example the result of zero divided by zero.
- `Inf` (Infinity). Represent positive and negative infinite values. For example the result of trying to divide any value by zero.

These values are converted from the R language representation to the SAS language representation as follows:

R Value	WPS Value
NA	. (missing value)
NaN	. (missing value)
+Inf	.I
-Inf	.M

Date values

Integer or real variables in the R language that have a `Date` class are formatted as `DATE9.` when imported into WPS.

R language `Date` values represent a count of days from the Unix epoch of 1 January 1970 UTC. Imported variables are adjusted to take account of the SAS language epoch of 1 January 1960.

Datetime values

Real variables in the R language that have a `POSIXct` class are formatted as `DATETIME19.` when imported into WPS.

R language `POSIXct` values represent a count of seconds from the Unix epoch of 1 January 1970 UTC. Imported variables are adjusted to take account of the SAS language epoch of 1 January 1960 and also use the value specified in the `GMTOFFSET` option of the `PROC R` statement.

Real variables in the R language that have a `times` class are formatted as `TIME8.` when imported into WPS.

Character values

On import, WPS scans a character variable and assigns a format that is the length of the longest string. Individual values in a character variable that are `NA` (Not Available) are converted to the SAS language missing character value (`' '`).

Factor values

An R language *factor* is a form of integer variable, where the values index a list of categorical variables (the list is known as a level in the R language). When imported into WPS these are converted into character variables in the dataset. The variable is given a length equal to the longest string in the levels list.

Using R graphics

When launching an R session, WPS configures R so that any graphics generated with the default graphics device are captured and included in the WPS session's standard ODS HTML output.

The following program extends the previous example to include simple linear regression analysis and graphics.

1. Create a new program file, paste the following code, and save the file:

```
data source;
  do x=1 to 10;
    y=ranuni(-1);
    output;
  end;

PROC R;
  export data=source;
  submit;
    model <- lm(source$y ~ source$x)
    print(model)
    par(mfrow=c(2, 2))
    plot(model)
  endsubmit;
run;
```

2. Run the program by clicking on the toolbar **Run** icon, and examine the HTML output.

The output includes the printed R results together with a single graphic generated within the R session and routed into the WPS output.

```
Call:
lm(formula = source$y ~ source$x)
Coefficients:
(Intercept)      source$x
    0.5344         0.0241
```

Using additional R packages

To use additional packages that are not included in your R installation, we recommend you install and check the basic operation of these packages in the interactive R environment. Installed packages can be used in an R program using the `library()` function.

An R environment launched by WPS inherits the environment variables from the WPS process. If third-party software is installed for use in R that requires, for example, additional entries in the *PATH* environment variable, Workbench must be restarted to register the changes.

SAS language macro processing

Using SAS language macros with in-line R programs.

When an in-line R program is run, the code between the `SUBMIT` and `ENDSUBMIT` statements is passed verbatim to the R interpreter. Macro processing is, therefore, suspended between the `SUBMIT` and `ENDSUBMIT` statements because:

- The R language uses the `&` and `%` characters as part of its syntax. Attempting to macro process the R source code might result in valid R syntax being misinterpreted as SAS language macro statements.

- The R language allows line-end style comments. The contents might contain, for example, unmatched apostrophes; tokenising of the R syntax using the regular SAS language parsing rules could not then occur.

Example

Demonstrates how to use a SAS language dataset in the R procedure to create a scatter plot diagram.

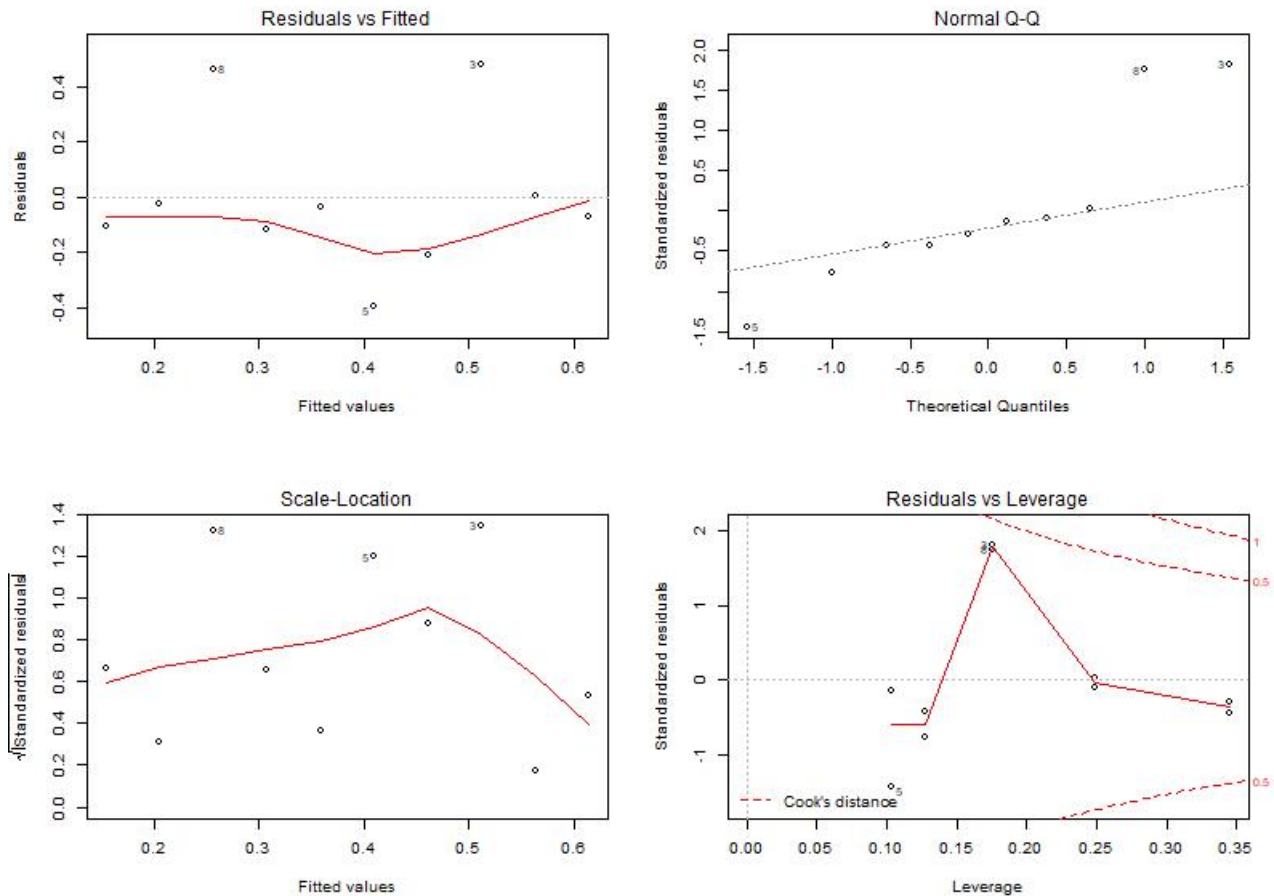
The following example creates a dataset in a SAS language `DATA` step, and then uses the `EXPORT` statement to pass that dataset to the R environment. The dataset is converted to a `data.frame` as part of the export, and the `data.frame` is used to create plots in a grid two plots wide and two plots deep.

An output PDF file destination is created using the SAS language Output Delivery System (ODS). Adding PDF to the output destinations includes the printed `data.frame` content and returned plot image file in the PDF output. The PDF is saved and the output can be viewed in a PDF viewer.

```
ODS PDF FILE='scatter_plot.pdf';
DATA SOURCE;
  DO X=1 TO 10;
    Y=RANUNI(-1);
    OUTPUT;
  END;
RUN;

PROC R;
  EXPORT DATA=source;
  SUBMIT;
    str(source)
    print(source)
    model <- lm(source$Y ~ source$X)
    print(model)
    par(mfrow=c(2, 2))
    plot(model)
    x <- (1:10)
  ENDSUBMIT;
  IMPORT R=x;
RUN;
ODS PDF CLOSE;
```

This creates the following plot in the ODS PDF output:



R procedure reference

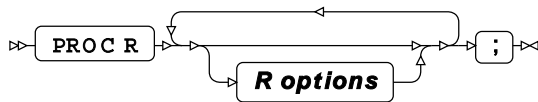
Describes the syntax and options for `PROC R` and its contained statements.

<code>PROC R</code> ↗	2445
Invokes the R environment that enables the execution of in-line or external R language programs.	
<code>ASSIGN</code> ↗	2447
The <code>ASSIGN</code> statement can be used to assign SAS language variable values to an R vector.	
<code>ENDSUBMIT</code> ↗	2448
Specifies the end of an in-line R language program.	
<code>EXECUTE</code> ↗	2449
Runs an R program stored in a separate file.	
<code>EXPORT</code> ↗	2450
Enables a SAS language dataset to be converted to an R <code>data.frame</code> and referenced in an R program.	

IMPORT ↗	2451
Enables an R language <code>data.frame</code> to be converted to a SAS language dataset and referenced in a SAS language program.	
LOAD ↗	2452
The <code>LOAD</code> statement deserialises an <i>R object</i> stored in a SAS language catalog.	
SAVE ↗	2453
Enables R objects to be serialised and stored in a SAS language catalog.	
SUBMIT ↗	2455
Specifies the start of an in-line R language program.	

PROC R

Invokes the R environment that enables the execution of in-line or external R language programs.



Datasets created in WPS can be made available to the R program using the `EXPORT` statement, and a dataset imported from the R program into WPS using the `IMPORT` statement.

An R program can be either written in-line in the R procedure, or run from a separate file:

- To run an in-line R program, use the `SUBMIT` and `ENDSUBMIT` statements.
- To run an R program stored in an external file use the `EXECUTE` statement.

The R environment is exited using a `RUN` statement.

Options

The following options are available.

GMTOFFSET

Specifies the offset to UTC applied when moving datasets between the SAS language and R language environments to take account the current time zone.

➤ GMTOFFSET = "+/-HH:MM" ➤

Date and time values in the R language are represented in UTC (Coordinated Universal Time) with an associated time zone. In the SAS language date and time values have no implied time zone. The specified `GMTOFFSET` is applied when using the `ASSIGN`, `EXPORT`, or `IMPORT` statements.

KEEP

Specifies that the current R environment is not terminated when the procedure exits.

A diagram showing the **KEEP** statement. It consists of a box with the word **KEEP** inside, flanked by two right-pointing arrows on the left and two left-pointing arrows on the right.

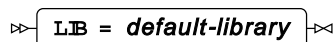
When specified, the current R environment is kept active when the current procedure exits, and the environment is used in the next invocation of the R procedure in the same program. If that invocation does not specify **KEEP**, the environment is terminated when the procedure exits.

The default behaviour is to terminate the R environment at the end of the procedure. Specifying **KEEP** keeps the current R environment, including any modules loaded during the execution of a R program, to be used in the next invocation of **PROC R**.

You can specify the **RKEEP** system option to use the same R environment for the duration of the execution of the SAS language program.

LIB

Specifies the default library location for the procedure step. The default location is the **WORK** library.

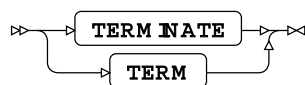
A diagram showing the **LIB** statement. It consists of a box containing the text **LIB = default-library**, flanked by two right-pointing arrows on the left and two left-pointing arrows on the right.

The **LIB** location is used:

- When exporting a dataset, and *libname* is not specified as part of the path for the **DATA** option of the **EXPORT** statement.
- When importing a dataset, and *libname* is not specified as part of the path for the **DATA** option of the **IMPORT** statement.
- When saving an R object to a SAS language catalog, and *libname* is not specified as part of the path for the **CATALOG** option of the **SAVE** statement.
- When loading an R object to a SAS language catalog, and *libname* is not specified as part of the path for the **CATALOG** option of the **LOAD** statement.

TERMINATE

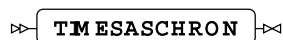
Specifies that the R environment is terminated when the procedure exits.

A diagram showing the **TERMINATE** statement. It consists of a box with the text **TERM NATE** inside, flanked by two right-pointing arrows on the left and two left-pointing arrows on the right. Below this box is another box with the text **TERM** inside, also flanked by two right-pointing arrows on the left and two left-pointing arrows on the right. A curved arrow points from the bottom of the **TERM NATE** box to the top of the **TERM** box.

Specifying **TERMINATE** stops the current R environment even if the **RKEEP** system option has been specified.

TIMESASCHRON

Specifies whether time values are represented in R using the **chron** class.

A diagram showing the **TIMESASCHRON** statement. It consists of a box with the text **TIMESASCHRON** inside, flanked by two right-pointing arrows on the left and two left-pointing arrows on the right.

By default, time values are stored in the R **POSIXct** type, representing a count of seconds from midnight. When **TIMESASCHRON** is specified, time values are stored in R as **chron.times** types.

To use the `TIMESASCHRON` option, you must include the `chron` package in your R environment using the `R library()` statement.

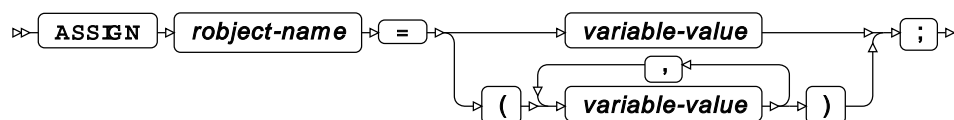
Example

The following example shows how to use `PROC R` to find the version of the R interpreter used with WPS. Version information is written to ODS output.

```
PROC R;  
  SUBMIT;  
    print(R.version)  
  ENDSUBMIT;  
RUN;
```

ASSIGN

The `ASSIGN` statement can be used to assign SAS language variable values to an R vector.



The `ASSIGN` statement is used to pass parameters to an R program, and the specified *object-name* can be used in an in-line program or a program run using the `EXECUTE` command. *object-name* is case sensitive and, unlike SAS language variables, must be referred to in the R program using the same case as in the `ASSIGN` definition.

Variables can be generated using SAS language macro variable expansion or execution. This enables you to preprocess variables using the SAS language and pass the result to an R program.

object-name

Specifies one or more variable values to pass to an R program. The *object-name* specifies the variable name by which the *variable-value* is referenced in the R program.

The *object-name* can be defined using a SAS language name literal (`'object.name'N`) to create an R object that would not be valid in the SAS language. For example, to assign a value Peter to an R object `employee.firstname`, the `ASSIGN` statement would be:

```
ASSIGN 'employee.firstname'N = 'Peter';
```

Multiple *variable-values* can be assigned to a single R language object. In this case, all values must be of the same type and in a comma-separated list in parenthesis.

Passing multiple variables to R

In this example, multiple vector variables are passed from a SAS language program to an R program that converts the vectors into a data frame.

```
PROC R;  
  ASSIGN Nu = (1, 2, 3, 4, 5);  
  ASSIGN Ch = ('Cyan', 'Magenta', 'Yellow', 'Black', 'Green');  
  SUBMIT;  
    DFrame <- data.frame(Nu, Ch)  
    print (DFrame)  
  ENDSUBMIT;  
RUN;
```

This produces the following output:

Nu	Ch
1	Cyan
2	Magenta
3	Yellow
4	Black
5	Green

Assigning a SAS language macro variable to an R object

In this example, the SAS language macro variable *PARM* is passed to an R program to determine the sample size in a randomly-generated sample.

```
%LET PARM=15;  
PROC R;  
  ASSIGN parm=&PARM;  
  SUBMIT;  
    x<-sample(1:3, parm, replace=TRUE)  
    print(x);  
  ENDSUBMIT;  
RUN;
```

This produces the following output:

```
[1] 1 1 3 3 3 2 1 1 3 1 1 3 3 1 1
```

ENDSUBMIT

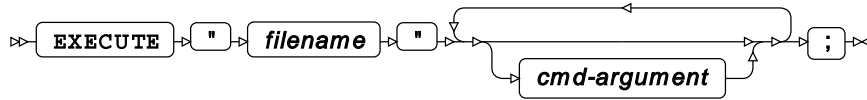
Specifies the end of an in-line R language program.

» **ENDSUBMIT** » ; »

The `ENDSUBMIT` statement must be entered at the start of a new line after the R language program.

EXECUTE

Runs an R program stored in a separate file.



The EXECUTE statement is an alternative to using the SUBMIT statement. It allows the R code to be placed in a separate file and enables you to run the same program in both WPS and an interactive R environment.

filename

A quoted string containing the path of the R program file. *filename* can be either an absolute path or a relative path.

When using Workbench to run an R language program, if a relative path is specified the root of the path is the Workspace. For example, to run a file named `math.r` from a project named `calculate`, the relative path is `calculate/math.r`.

cmd-argument

Specifies a command line argument passed to the R program.

An example of executing an R program stored in a file

In this example, an R program stored in an external file `model.r` is executed in the R procedure. The contents of `model.r` source file:

```
model <- lm(source$Y ~ source$X)
print(model)
par(mfrow=c(2, 2))
plot(model)
```

The following program creates a dataset. The dataset is passed to the R program using the EXPORT statement before the program is run using the `model.r` file:

```
DATA SOURCE;
  DO X=1 TO 10;
    Y=RANUNI(-1);
  OUTPUT;
END;

PROC R;
  EXPORT DATA=source;
  EXECUTE "model.r";
RUN;
```

EXPORT

Enables a SAS language dataset to be converted to an R `data.frame` and referenced in an R program.



Export options

The following options are available with the `EXPORT` statement.

DATA

Specifies the name of the WPS dataset.

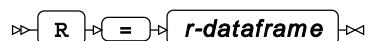


The library location can be specified using either *libname* in the `DATA` option, or the `LIB` option of the `PROC R` statement:

- If *libname* is specified, that location is always used.
- If the `LIB` option of the `PROC R` statement is specified and *libname* is not specified, the location in the `LIB` option is used.
- If neither *libname* or the `LIB` option on the `PROC R` statement are specified, the `WORK` location is used.

R

Specifies the name of the `data.frame` as used in the R environment.



If this option is not specified, the *r-dataframe* default is the *dataset* name specified in the `DATA` option. If you use the default *dataset* name in an in-line R language program, the variable name must match the case used in the `DATA` option.

An example of exporting data from WPS to R

The following example creates a dataset containing two numeric columns. The dataset is exported to the R environment and the content of the `data.frame` printed to ODS output.

```
DATA SOURCE;  
  DO X=1 TO 10;  
    Y=RANUNI(-1);  
    OUTPUT;  
  END;  
  
PROC R;  
  EXPORT DATA=SOURCE;  
  SUBMIT;  
    str(source)  
  ENDSUBMIT;  
RUN;
```

This produces the following output:

```
'data.frame': 10 obs. of 2 variables:  
 $ x: num  1 2 3 4 5 6 7 8 9 10  
 $ y: num  0.371 0.924 0.59 0.434 0.962 ...
```

IMPORT

Enables an R language `data.frame` to be converted to a SAS language dataset and referenced in a SAS language program.

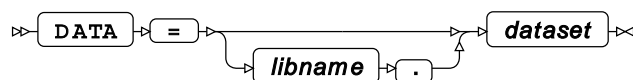


Import options

The following options are available with the `IMPORT` statement.

DATA

Specifies the dataset location and name as used in the WPS Analytics SAS language environment.



The library location can be specified using either `libname` in the `DATA` option, or the `LIB` option of the `PROC R` statement:

- If `libname` is specified, that location is always used.
- If the `LIB` option of the `PROC R` statement is specified and `libname` is not specified, the location in the `LIB` option is used.

- If neither *libname* nor the `LIB` option on the `PROC R` statement are specified, the `WORK` location is used.

If this option is not specified, the *dataset* default is the *r-dataframe* name specified in the `R` option.

R

Specifies the name of the `data.frame` as used in the R environment. Must be specified



r-dataframe is case sensitive and must match the case used for the imported `data.frame` in the R program.

LOAD

The `LOAD` statement deserialises an *R object* stored in a SAS language catalog.



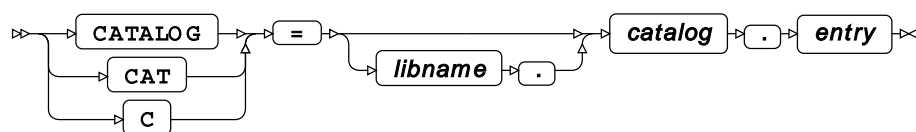
The `SAVE` and `LOAD` statements enable an R object to be serialised and stored in a SAS language catalog and later deserialised in a WPS session. The `SAVE` statement serialises an R object and stores it in an entry in a catalog. The `LOAD` statement deserialises an R object from a catalog.

Load options

The following options are available with the `LOAD` statement.

CATALOG

Specifies the catalog from which the stored R object is loaded.



A SAS language catalog is defined using:

libname

Specifies the name of the library in which the catalog is stored. The library location can be specified using either *libname* or the `LIB` option of the `PROC R` statement.

- If *libname* is specified, that location is always used.
- If the `LIB` option of the `PROC R` statement is specified and *libname* is not specified, the location in the `LIB` option is used.
- If neither *libname* nor the `LIB` option on the `PROC R` statement are specified, the `WORK` location is used.

catalog

Specifies the name of the catalog.

entry

Specifies the name of the R object in the catalog.

R

Specifies the variable name for the loaded R object as used in the R language program.



Because R is case-sensitive, *object-name* must match the case used of the R variable name. The name can be specified using name literal syntax (for example "r.object.name"N) if the name of the R object does not follow the normal rules for identifiers in the SAS language.

An example of using the **LOAD** statement

```
proc r;
  load cat=catalog.entry r='target.object'n;
run;
```

SAVE

Enables R objects to be serialised and stored in aSAS language catalog.



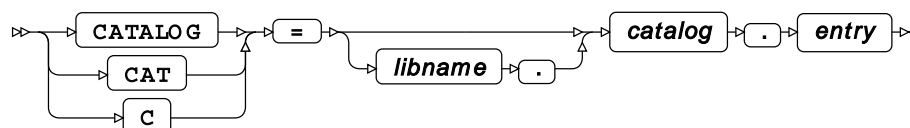
The **SAVE** and **LOAD** statements enable an R object to be serialised and stored in a SAS language catalog and later deserialised in a WPS session. The **SAVE** statement serialises an R object and stores it in an entry in a catalog. The **LOAD** statement deserialises an R object from a catalog.

Save options

The following options are available with the **SAVE** statement.

CATALOG

Specifies the location in which the R object will be saved.



The library location can be specified using either *libname* or the **LIB** option of the **PROC R** statement:

libname

Specifies the name of the library in which the catalog is stored. The library location can be specified using either *libname* or the `LIB` option of the `PROC R` statement.

- If *libname* is specified, that location is always used.
- If the `LIB` option of the `PROC R` statement is specified and *libname* is not specified, the location in the `LIB` option is used.
- If neither *libname* nor the `LIB` option on the `PROC R` statement are specified, the `WORK` location is used.

catalog

Specifies the name of the catalog.

entry

Specifies the name of the R object in the catalog.

R

Specifies the name of the R object to save to the catalog. Must be specified.

➤ `R` ➤ `=` ➤ `object-name` ➤

Because `R` is case-sensitive, the case of *object-name* must match the object name in the `R` program. The name can be specified using name literal syntax (for example `"r.object.name"`) if the name of the `R` object does not follow the normal rules for identifiers in the SAS language.

DESCRIPTION

Specifies a description string saved with the catalog entry.

➤ `DESCRIPTION` ➤ `=` ➤ `"Catalog entry description"` ➤

The description is displayed in the output from the `PROC CATALOG` statement.

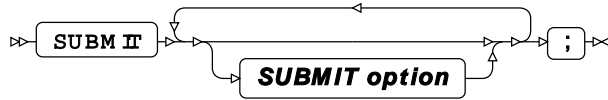
The catalog entry will have a type of `OBJECT`.

An example of saving an R object to a WPS catalog

```
proc r;  
  save cat=catalog.entry r='source.object'n;  
run;
```

SUBMIT

Specifies the start of an in-line R language program.



An in-line R language program is defined as part of the R procedure in a SAS language program. The `SUBMIT` statement marks the start of the program, and the `ENDSUBMIT` statement marks the end.

An R language program must start on a new line after the `SUBMIT` statement, and the `ENDSUBMIT` statement must appear at the beginning of a line on its own.

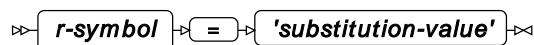
Multiple in-line R language programs can exist in a single R procedure environment. Each R language program is executed as it is encountered. Variables defined in one R language program can be used in subsequent in-line programs in the same R procedure environment.

SUBMIT options

The following option can be used with the `SUBMIT` statement.

r-symbol

Enables the replacement of a symbol in the R program with a string.



Before being passed to the R environment, an in-line R program is pre-processed to replace any *r-symbol* defined in the program with the content of the *substitution-value*. If the *substitution-value* is a string, it must be surrounded by quotation marks. If *substitution-value* is a SAS language macro variable, it must be prepended by an ampersand (&), but quotation marks are not required.

The *r-symbol* name is case sensitive and must be referred to in the R program using the same case as in the `SUBMIT` statement definition.

```
PROC R;  
SUBMIT greeting = 'Hello World';  
r.welcome <- "&greeting."  
print (r.welcome)  
ENDSUBMIT;  
RUN;
```

The syntax for replacing a defined *r-symbol* is the same as that used for SAS language macro variable substitution, the name of the defined *r-symbol* is prepended by an ampersand (&) in the inline code.

SUBMIT statement – basic example

The following example shows how to incorporate an R-language program into a SAS language program:

```
PROC R;  
SUBMIT;  
x <- (1:10)  
print(x)  
ENDSUBMIT;  
RUN;
```

Which outputs the following:

```
[1]  1  2  3  4  5  6  7  8  9 10
```

SUBMIT statement – using a macro variable in an R program

The following example shows how to pass a variable from a SAS language program to an R language program. The variable *welcome* is defined using the SAS language %LET macro. The specified macro variable is assigned to the *r-symbol* of the SUBMIT statement. The *r-symbol* is then referenced in the inline R language program:

```
%LET welcome = 'Hello World';  
PROC R;  
SUBMIT greeting = &welcome;  
r.welcome <- "&greeting"  
print (toupper(r.welcome))  
ENDSUBMIT;  
RUN;
```

Which outputs the following:

```
[1] "HELLO WORLD"
```

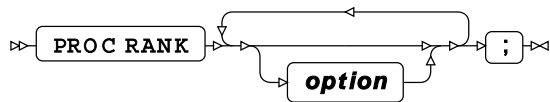
RANK procedure

Supported statements

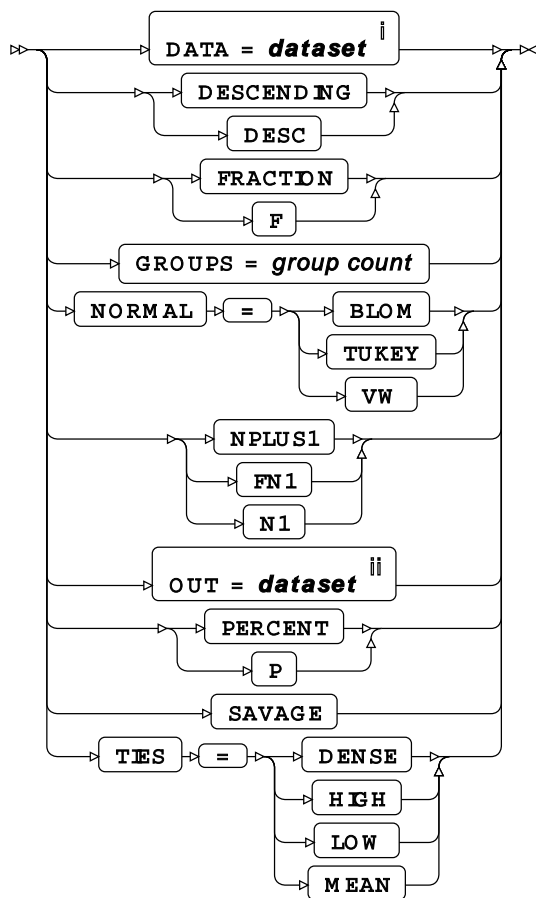
- *PROC RANK* [↗](#) (page 2457)
- *BY* [↗](#) (page 2458)
- *RANKS* [↗](#) (page 2458)
- *VAR* [↗](#) (page 2458)
- *WHERE* [↗](#) (page 2458)

PROC RANK

Assigns a rank that replaces the original value to every observation of one or more specified variables in a dataset.



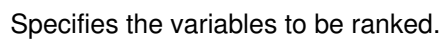
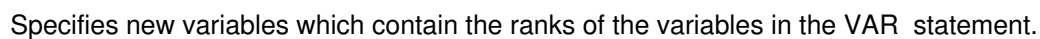
option



ⁱ See *Input dataset* [↗](#) (page 16).

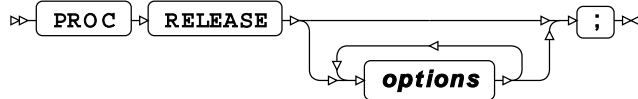
ⁱⁱ See *Output dataset* [↗](#) (page 16).

Groups the observations in a dataset using one or more specified variables.



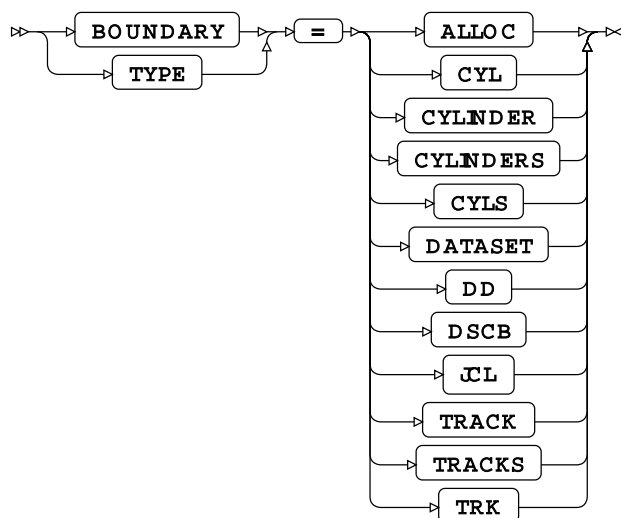
PROC RELEASE

Releases unused space in a Z/OS dataset, if possible.



Options

BOUNDARY



ALLOC

CYL

CYLINDER

CYLINDERS

CYLS

DATASET

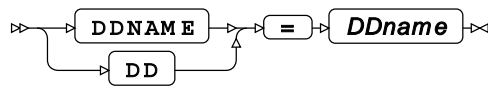
DD

DSCB

JCL

TRACK

TRACKS

TRK**DDNAME****TRACE**

Type: Keyword

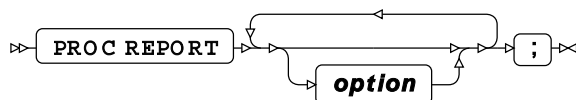
REPORT procedure

Supported statements

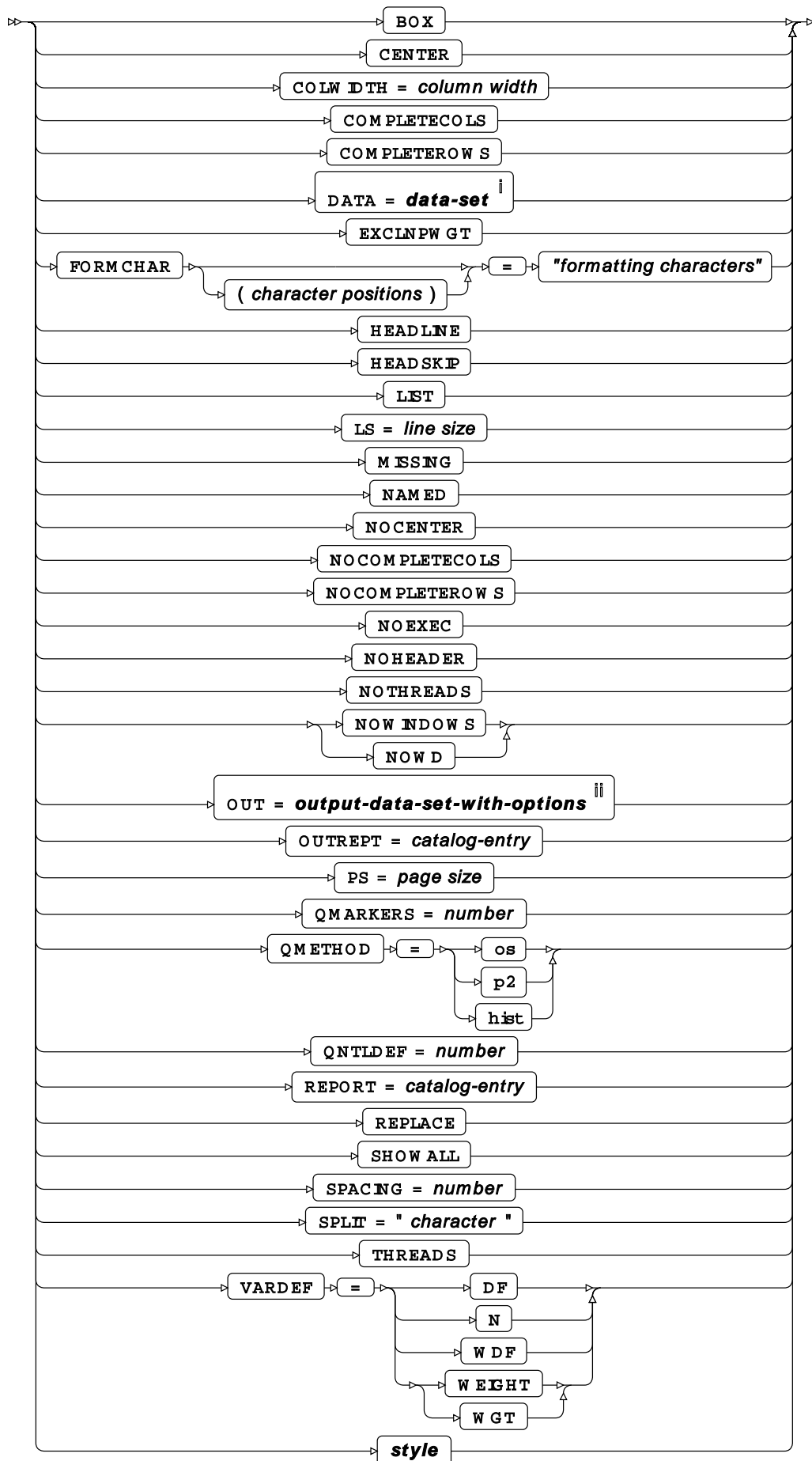
- *PROC REPORT* [↗](#) (page 2460)
- *BREAK* [↗](#) (page 2462)
- *BY* [↗](#) (page 2463)
- *COLUMN* [↗](#) (page 2463)
- *COMPUTE* [↗](#) (page 2465)
- *DEFINE* [↗](#) (page 2465)
- *FREQ* [↗](#) (page 2467)
- *RBREAK* [↗](#) (page 2468)
- *WEIGHT* [↗](#) (page 2468)

PROC REPORT

Produces statistical reports with options on formatting and outputting the dataset.

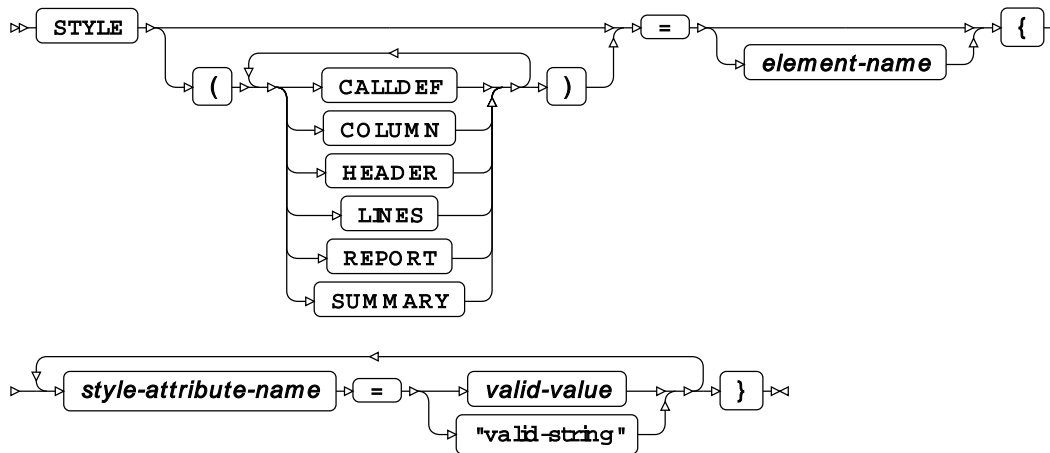


option



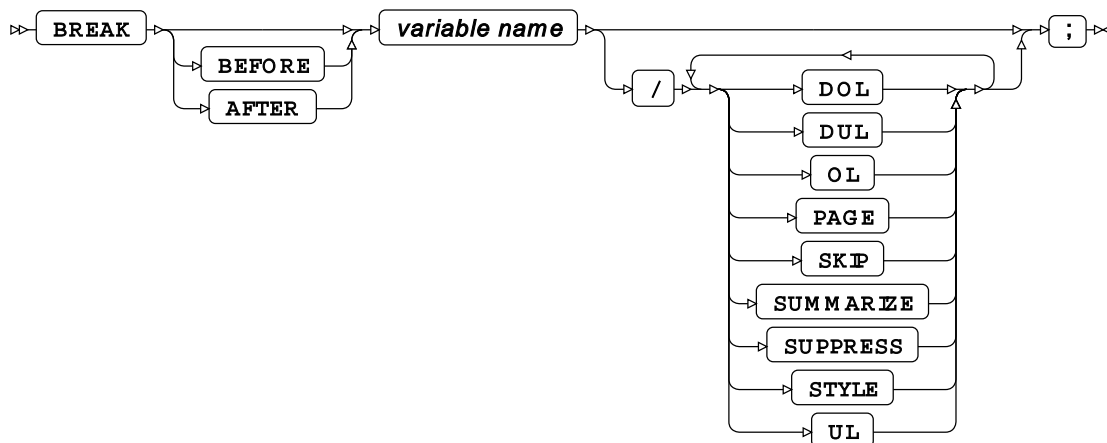
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).

style



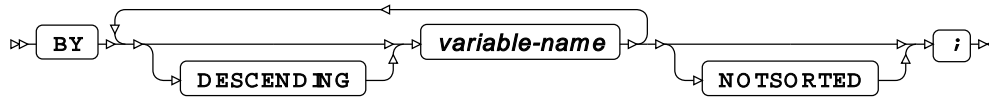
BREAK

Summarises information to be displayed before or after specific data has been reported.



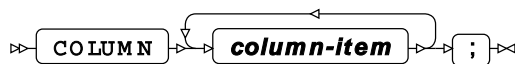
BY

Groups the observations in a dataset using one or more specified variables.

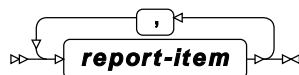


COLUMN

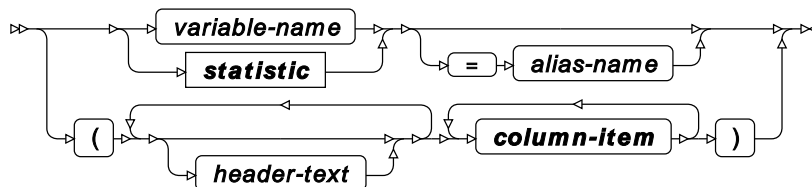
Determines the columns and headings in a report.



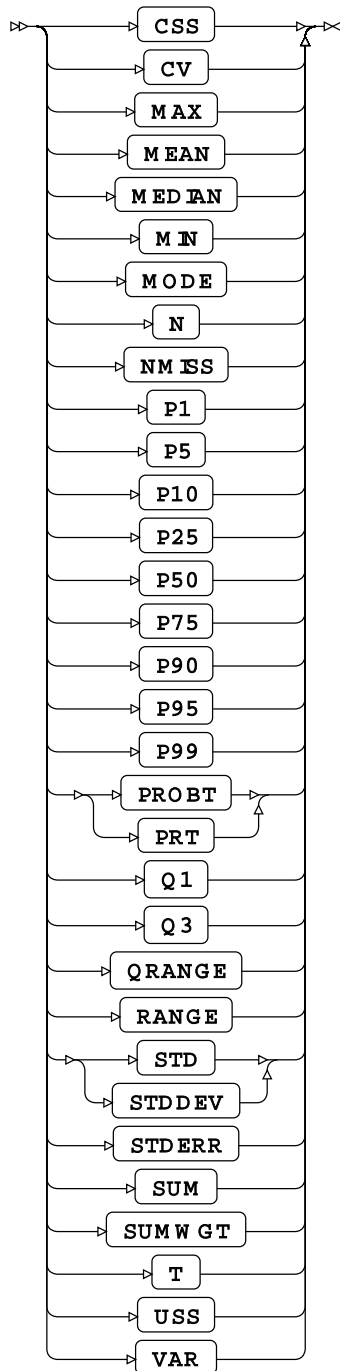
column-item



report-item

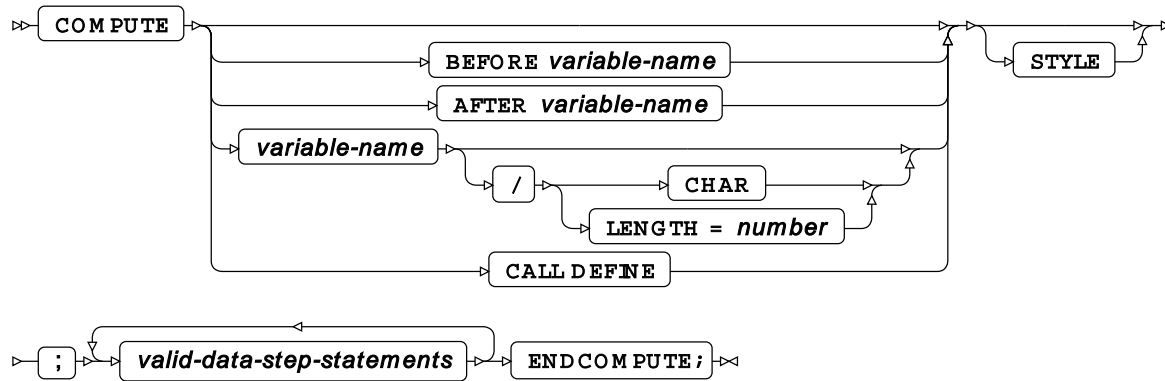


statistic



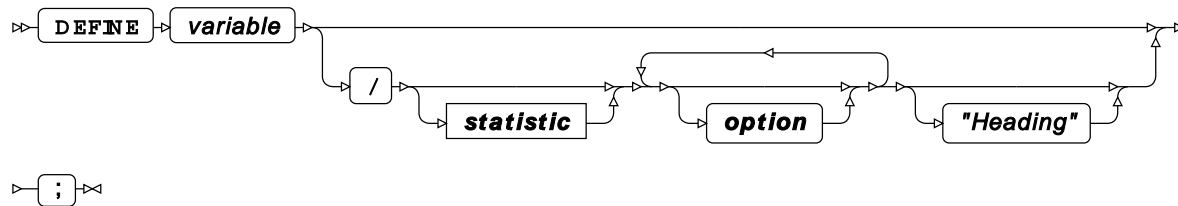
COMPUTE

Calculates statistics for each row before or after a given variable.

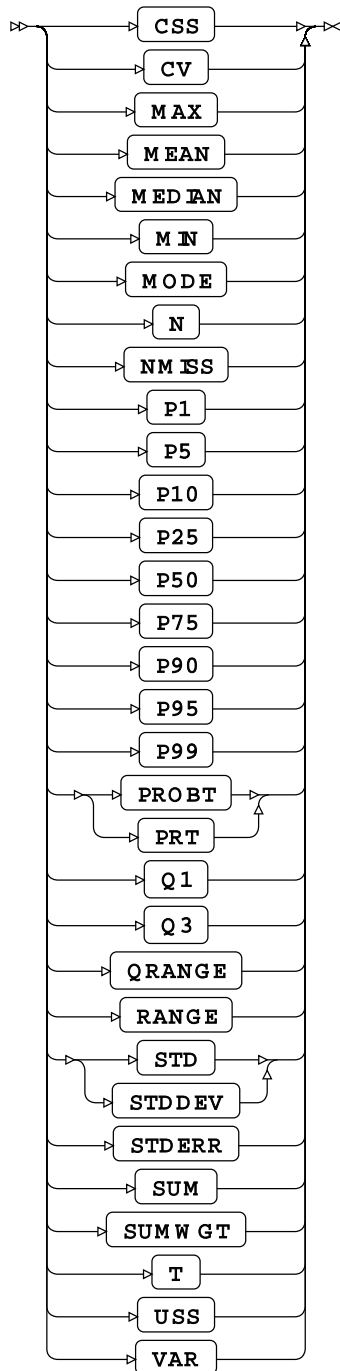


DEFINE

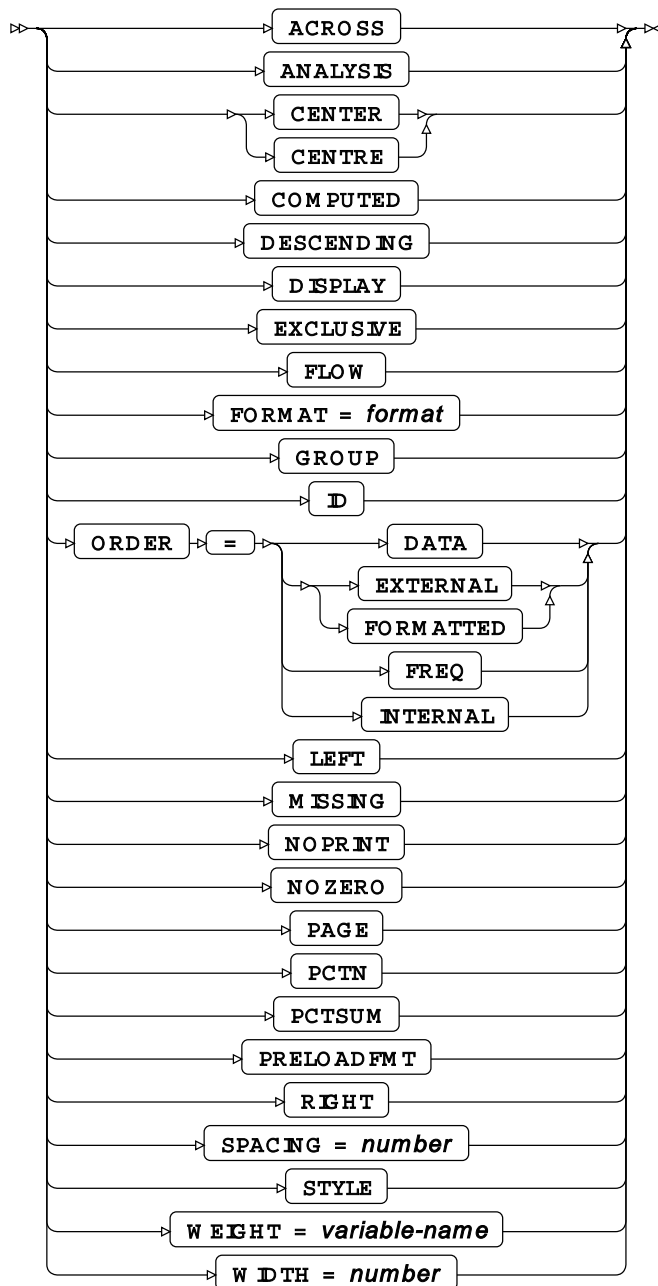
Defines the appearance of columns and variables used in the **COLUMN** statement.



statistic

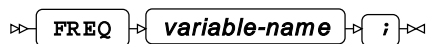


option



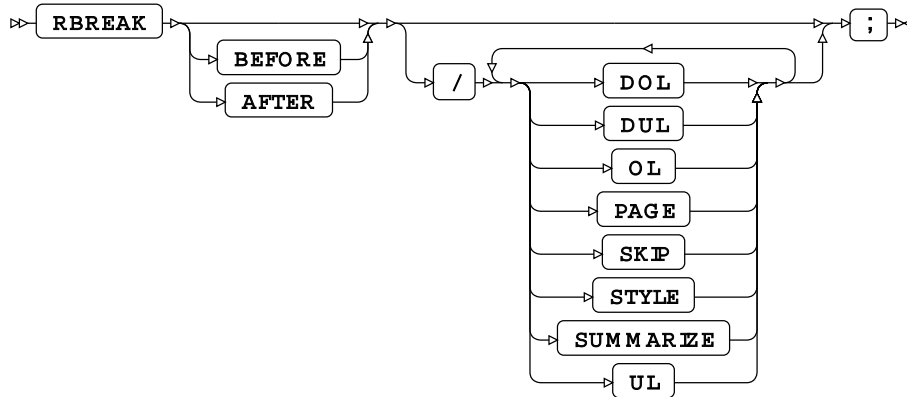
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



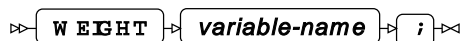
RBREAK

Summarises information to be displayed from the whole report.



WEIGHT

Specifies a variable giving the weight associated with each observation.



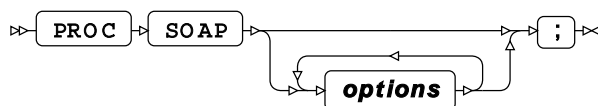
SOAP procedure

Supported statements

- *PROC SOAP* [↗](#) (page 2468)

PROC SOAP

Exchanges structured information with web services, by a protocol.



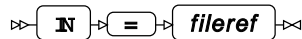
Options

ENVELOPE



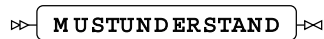
Type: Keyword

IN



Type: String

MUSTUNDERSTAND



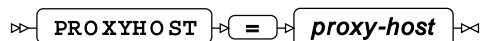
Type: Keyword

OUT



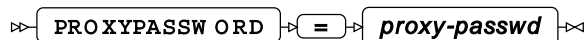
Type: String

PROXYHOST



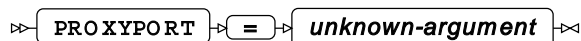
Type: String

PROXYPASSWORD

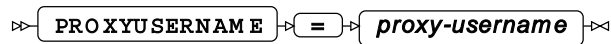


Type: String

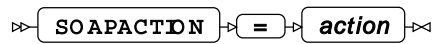
PROXYPORT



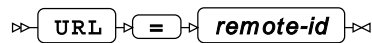
Type: Numeric

PROXYUSERNAME

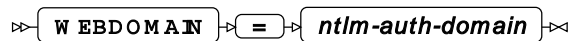
Type: String

SOAPACTION

Type: String

URL

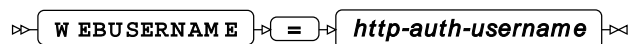
Type: String

WEBDOMAIN

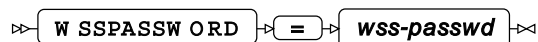
Type: String

WEBPASSWORD

Type: String

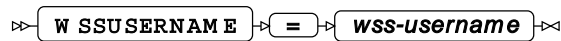
WEBUSERNAME

Type: String

WSSPASSWORD

Type: String

WSSUSERNAME



Type: String

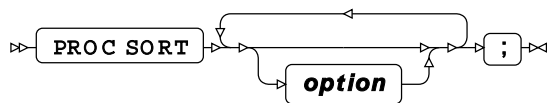
SORT procedure

Supported statements

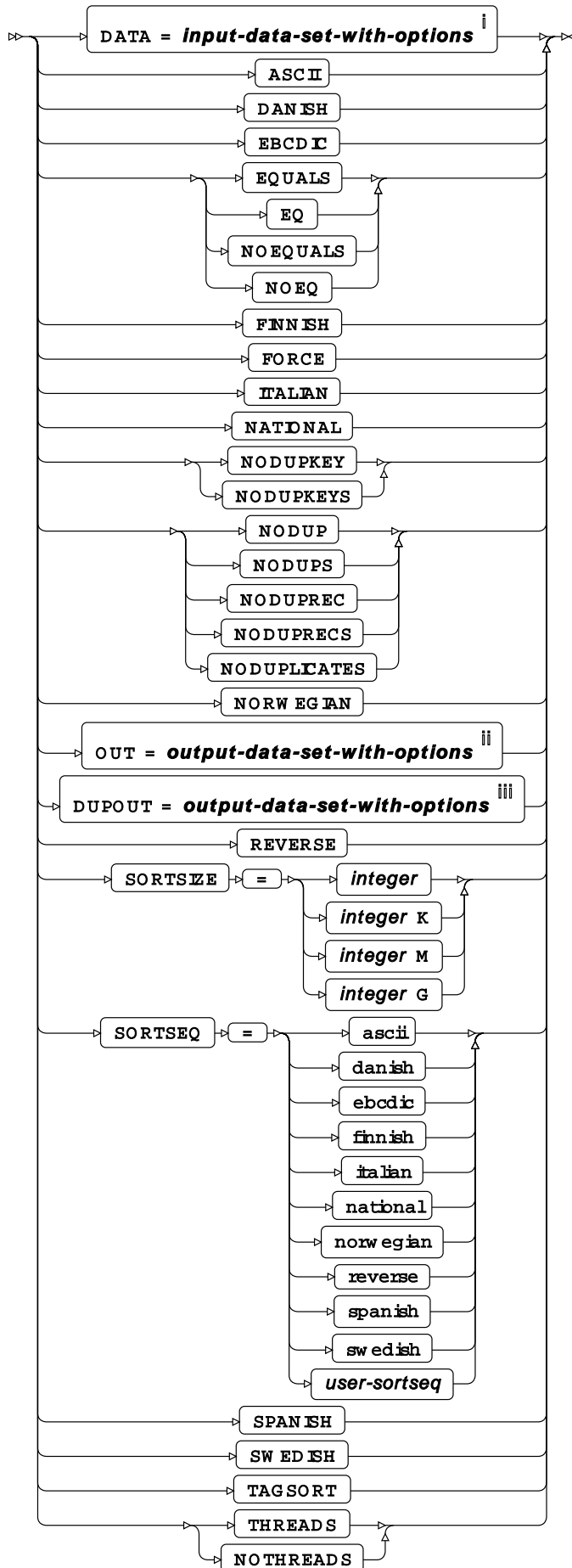
- *PROC SORT* [↗](#) (page 2471)
- BY [↗](#) (page 2473)
- WHERE [↗](#) (page 2473)

PROC SORT

Sorts an input dataset and writes to an output dataset.



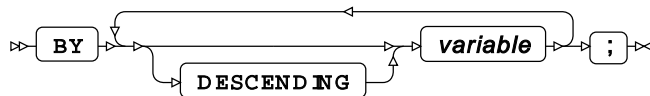
option



- i See *Input dataset* [↗](#) (page 16).
- ii See *Output dataset* [↗](#) (page 16).
- iii See *Output dataset* [↗](#) (page 16).

BY

Specifies the variable to sort by, and specifies the sorting order.



WHERE

Restricts the observations to be processed.



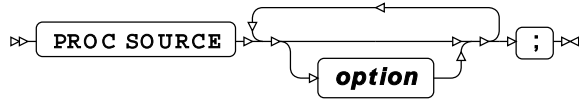
SOURCE procedure

Supported statements

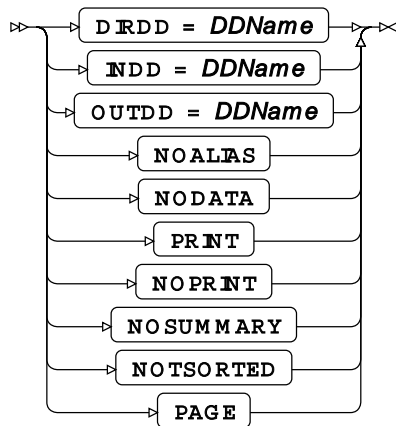
- *PROC SOURCE* [↗](#) (page 2474)
- *SELECT* [↗](#) (page 2474)
- *EXCLUDE* [↗](#) (page 2474)
- *FIRST* [↗](#) (page 2475)
- *LAST* [↗](#) (page 2475)
- *AFTER* [↗](#) (page 2475)
- *BEFORE* [↗](#) (page 2475)

PROC SOURCE

Transforms items in a partitioned dataset into a stream of sequential records.

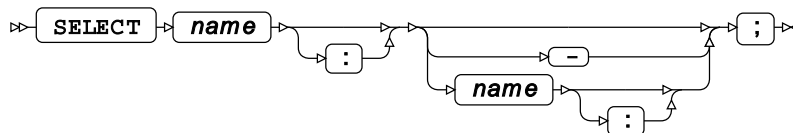


option



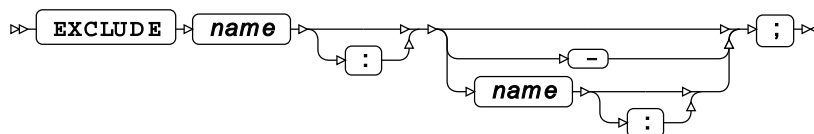
SELECT

Selects one or more names within a partitioned dataset, that may be processed. The colon (":") character is a wildcard.



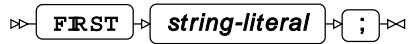
EXCLUDE

Excludes one or more names within a partitioned dataset, from being processed. The colon (":") character is a wildcard.



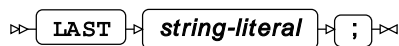
FIRST

Outputs specified text before any records.



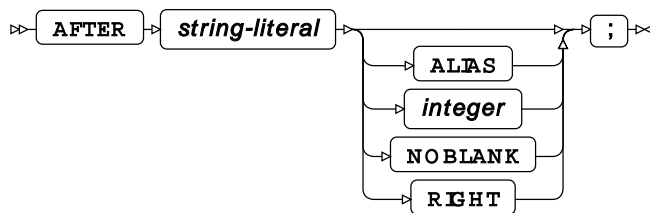
LAST

Outputs specified text after all records.



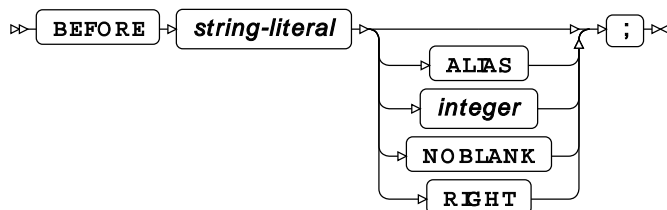
AFTER

Outputs specified text after this record.



BEFORE

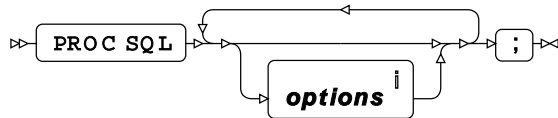
Outputs specified text before this record.



SQL Procedure

PROC SQL

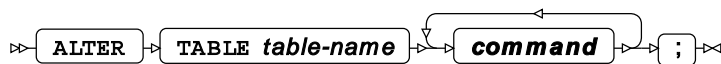
Enables the use of SQL statements in SAS language programs.



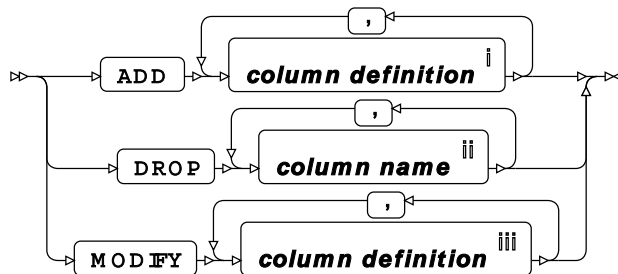
ⁱ See *PROC SQL options* [↗](#) (page 2482).

ALTER

Updates data held in a dataset or database table, or modifies the dataset or database table definition.



command



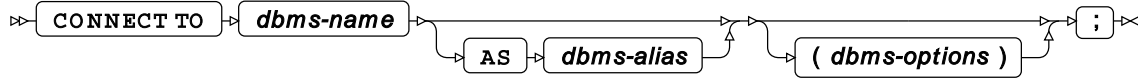
ⁱ See *Column Definition* [↗](#) (page 2493).

ⁱⁱ See *Column Name* [↗](#) (page 2494).

ⁱⁱⁱ See *Column Definition* [↗](#) (page 2493).

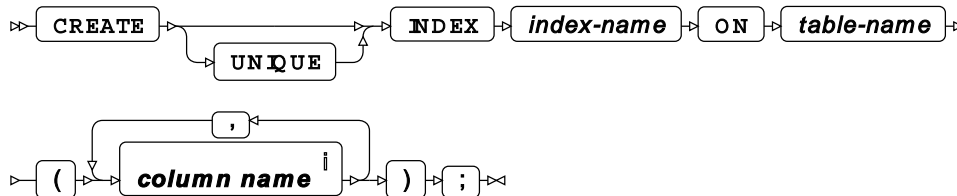
CONNECT

Creates a connection to a database server.



CREATE INDEX

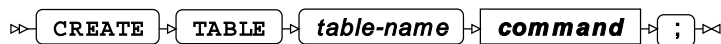
Creates an index from the specified columns in a dataset or database table to improve searching or data sorting.



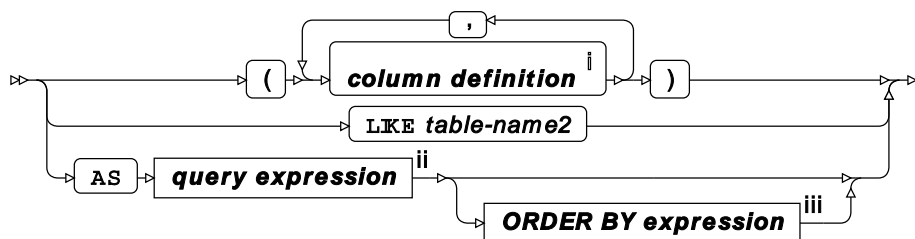
ⁱ See [Column Name](#) (page 2494).

CREATE TABLE

Creates a new dataset or table in a database, defining each column label and type.



command



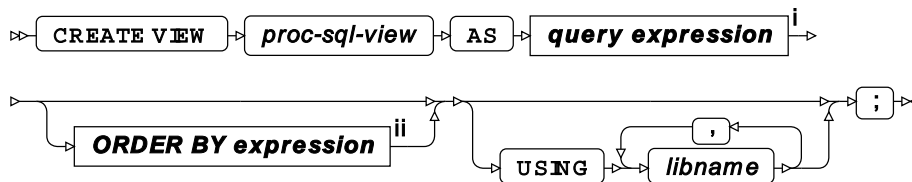
ⁱ See [Column Definition](#) (page 2493).

ⁱⁱ See [Query expression](#) (page 2490).

ⁱⁱⁱ See [ORDER BY expression](#) (page 2488).

CREATE VIEW

Creates a view, which defines a query against one or more datasets or database tables. The query is run whenever the view is used.

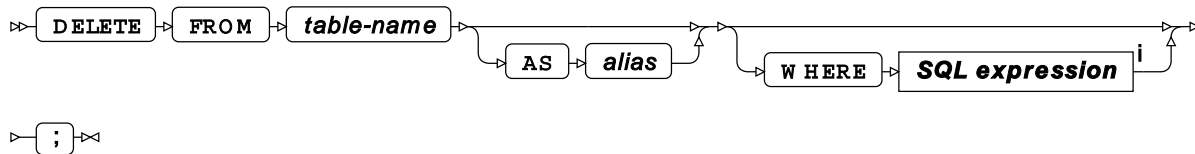


ⁱ See *Query expression* [↗](#) (page 2490).

ⁱⁱ See *ORDER BY expression* [↗](#) (page 2488).

DELETE

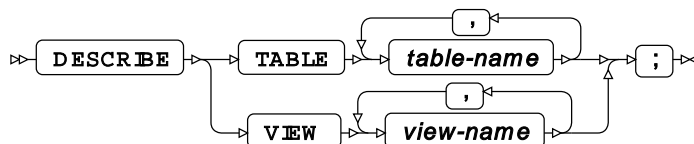
Deletes the specified rows from a dataset or database table.



ⁱ See *SQL expression* [↗](#) (page 2491).

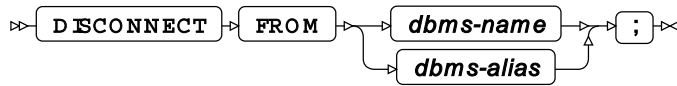
DESCRIBE

Shows the structure of a dataset, database table, or the query used to define a view.



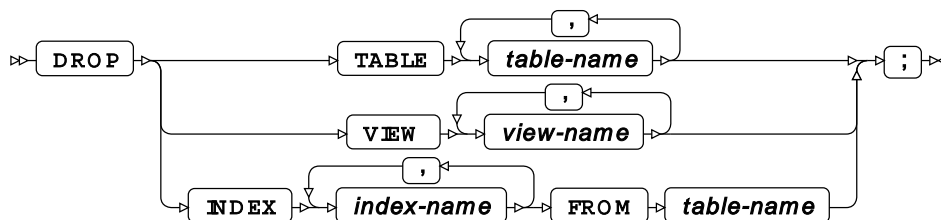
DISCONNECT

Closes the connection to a database server.



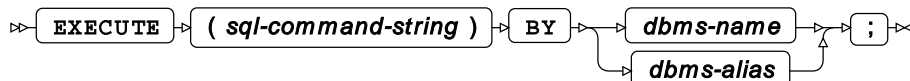
DROP

Deletes a dataset or database table, an index created for a dataset or database table, or the query used to create a view.



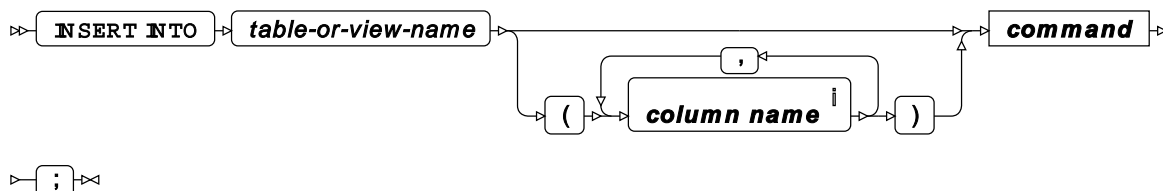
EXECUTE

Passes an SQL statement to a database server to be run directly on the server rather than in WPS. This statement can only be used when a `CONNECT-TO` connection has been created.



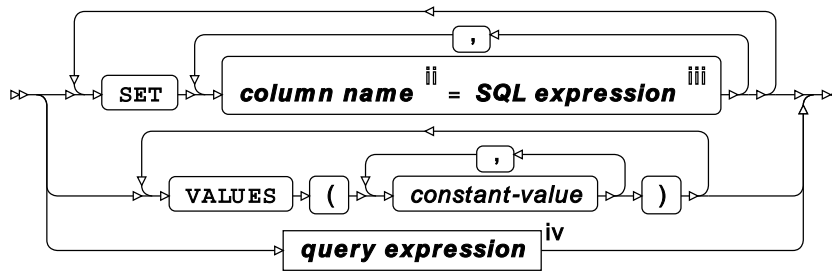
INSERT

Inserts data into a dataset a database table, or into an updateable view.



ⁱ See [Column Name](#) (page 2494).

command



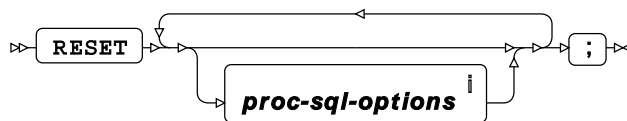
ⁱⁱ See [Column Name](#) (page 2494).

ⁱⁱⁱ See [SQL expression](#) (page 2491).

^{iv} See [Query expression](#) (page 2490).

RESET

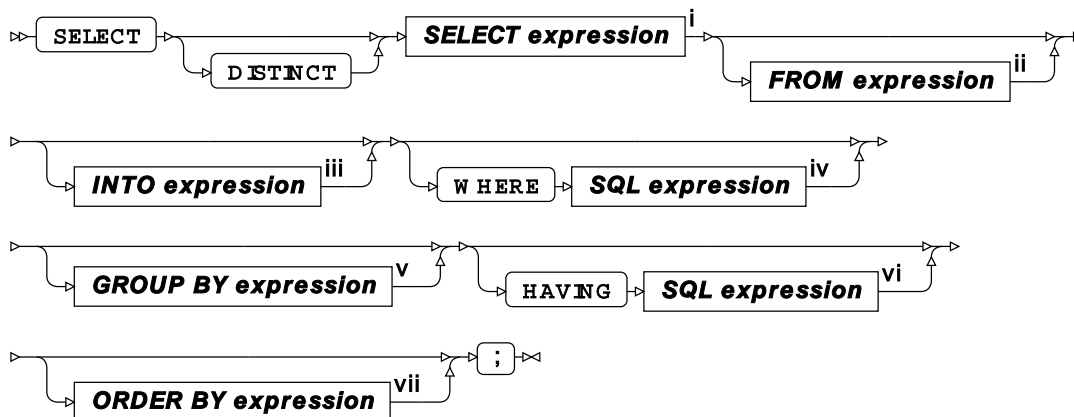
Provides reset options as part of executing a SQL script.



ⁱ See [PROC SQL options](#) (page 2482).

SELECT

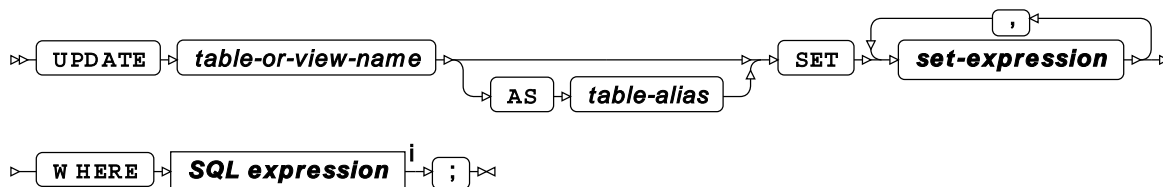
Retrieves information from a dataset, database table, or view.



- ⁱ See *SELECT expression* [↗](#) (page 2486).
- ⁱⁱ See *FROM expression* [↗](#) (page 2487).
- ⁱⁱⁱ See *INTO expression* [↗](#) (page 2487).
- ^{iv} See *SQL expression* [↗](#) (page 2491).
- ^v See *GROUP BY expression* [↗](#) (page 2488).
- ^{vi} See *SQL expression* [↗](#) (page 2491).
- ^{vii} See *ORDER BY expression* [↗](#) (page 2488).

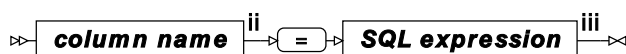
UPDATE

Updates a dataset, a database table, or an updateable view.



- ⁱ See *SQL expression* [↗](#) (page 2491).

set-expression



- ⁱⁱ See *Column Name* [↗](#) (page 2494).
- ⁱⁱⁱ See *SQL expression* [↗](#) (page 2491).

VALIDATE

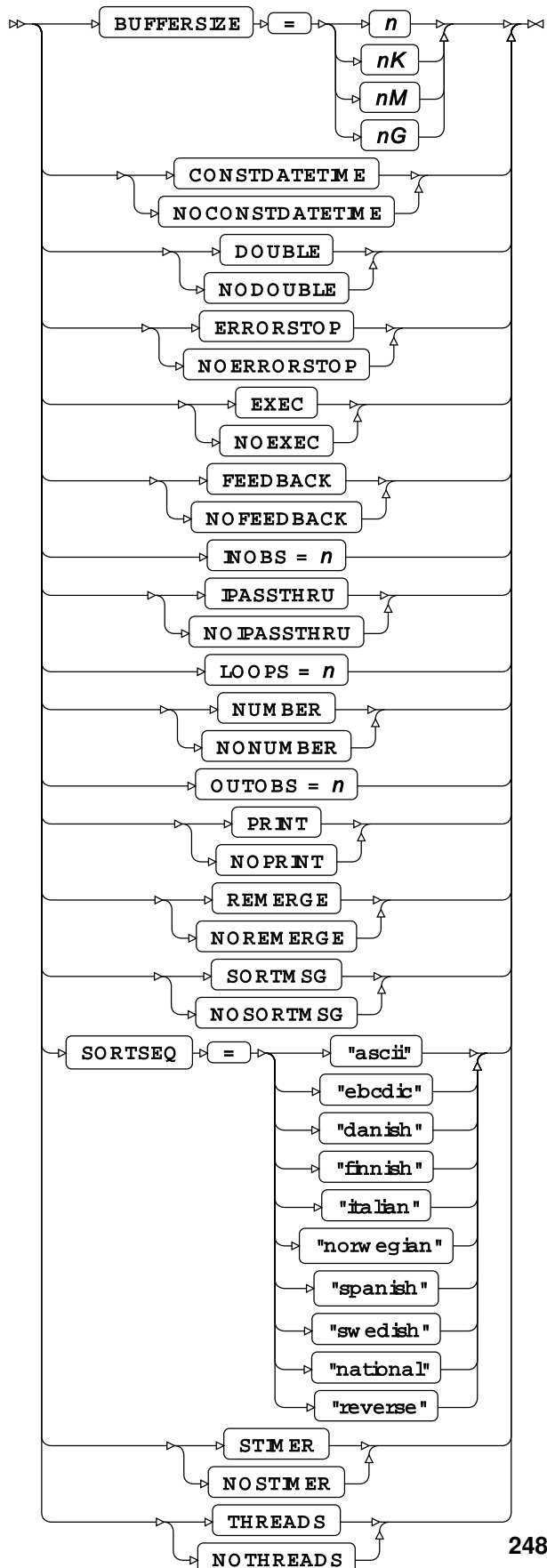
Checks that the SQL query is syntactically correct without running the statement.



- ⁱ See *Query expression* [↗](#) (page 2490).

Component Dictionary

PROC SQL options



Conditions

ALL condition



ⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱ See *Relational expression* [↗](#) (page 2489).

ⁱⁱⁱ See *Query expression* [↗](#) (page 2490).

ANY condition



ⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱ See *Relational expression* [↗](#) (page 2489).

ⁱⁱⁱ See *Query expression* [↗](#) (page 2490).

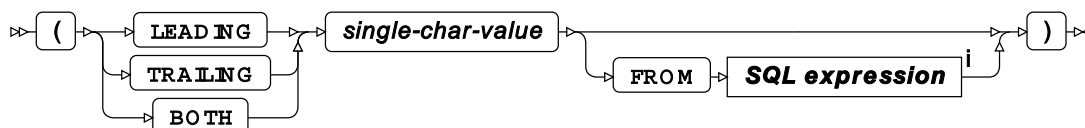
BETWEEN condition



ⁱ See *SQL expression* [↗](#) (page 2491).

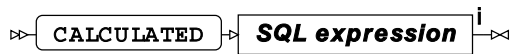
ⁱⁱ See *SQL expression* [↗](#) (page 2491).

BTRIM condition



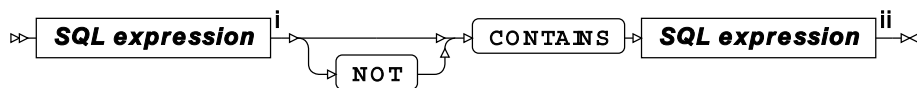
ⁱ See *SQL expression* [↗](#) (page 2491).

CALCULATED condition



ⁱ See *SQL expression* [↗](#) (page 2491).

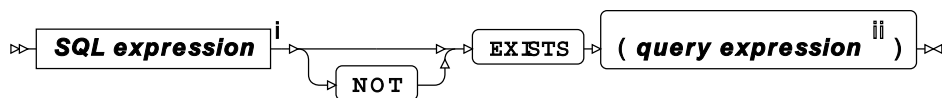
CONTAINS condition



ⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱ See *SQL expression* [↗](#) (page 2491).

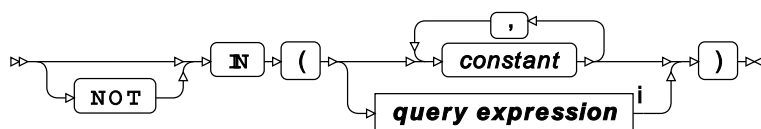
EXISTS condition



ⁱ See *SQL expression* [↗](#) (page 2491).

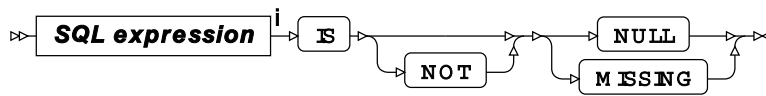
ⁱⁱ See *Query expression* [↗](#) (page 2490).

IN condition



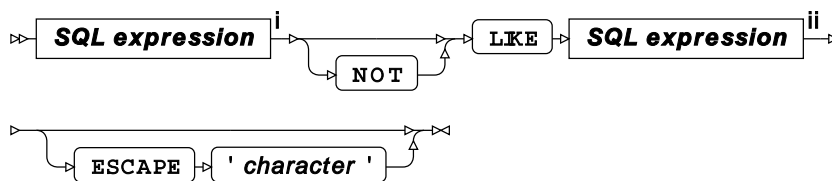
ⁱ See *Query expression* [↗](#) (page 2490).

IS condition



ⁱ See *SQL expression* [↗](#) (page 2491).

LIKE condition

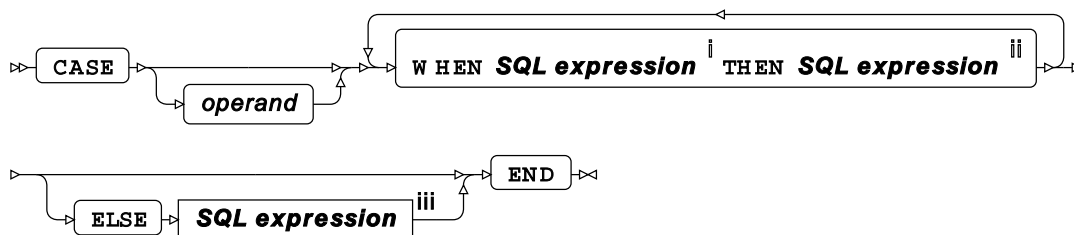


ⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱ See *SQL expression* [↗](#) (page 2491).

Expressions

CASE expression

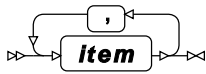


ⁱ See *SQL expression* [↗](#) (page 2491).

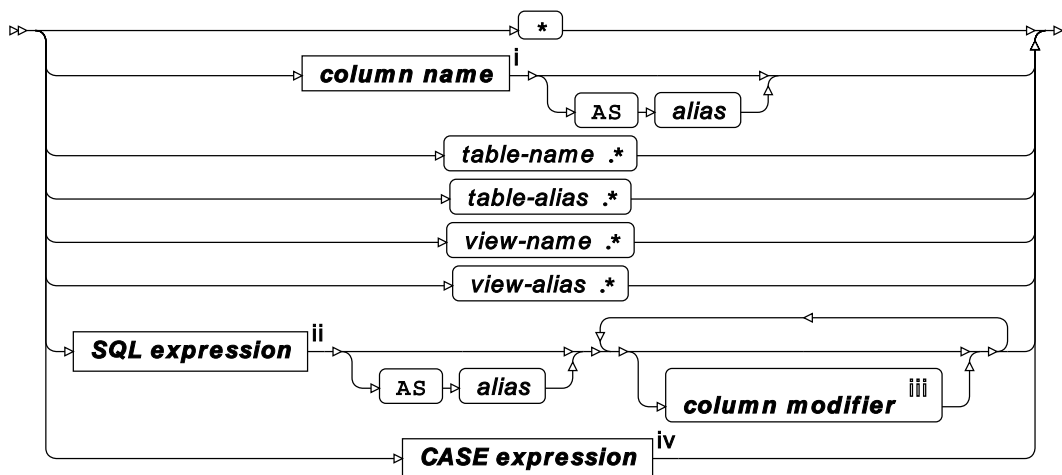
ⁱⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱⁱ See *SQL expression* [↗](#) (page 2491).

SELECT expression



item



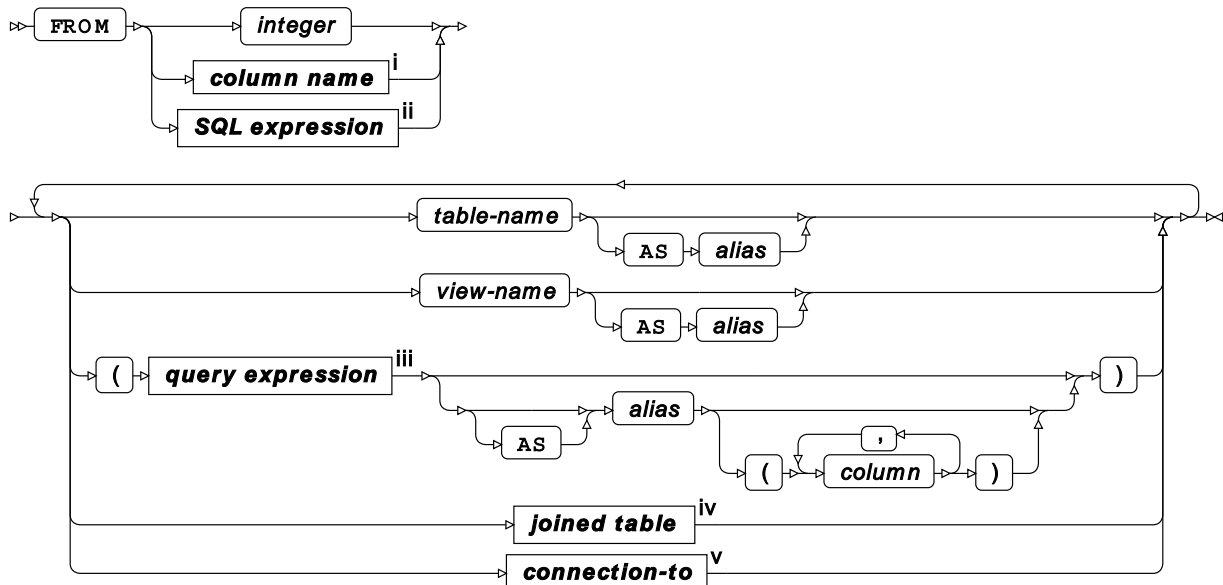
ⁱ See [Column Name](#) (page 2494).

ⁱⁱ See [SQL expression](#) (page 2491).

ⁱⁱⁱ See [Column Modifier](#) (page 2494).

^{iv} See [CASE expression](#) (page 2485).

FROM expression



ⁱ See [Column Name](#) (page 2494).

ⁱⁱ See [SQL expression](#) (page 2491).

ⁱⁱⁱ See [Query expression](#) (page 2490).

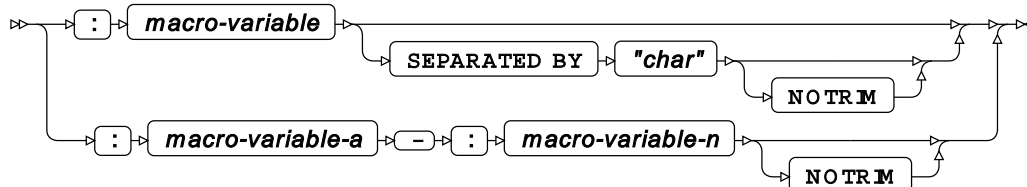
^{iv} See [Joined Table](#) (page 2494).

^v See [CONNECTION-TO](#) (page 2495).

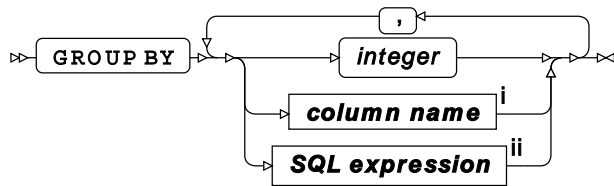
INTO expression



macro-variable-specification



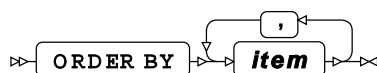
GROUP BY expression



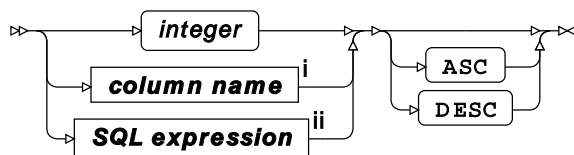
ⁱ See *Column Name* [↗](#) (page 2494).

ⁱⁱ See *SQL expression* [↗](#) (page 2491).

ORDER BY expression



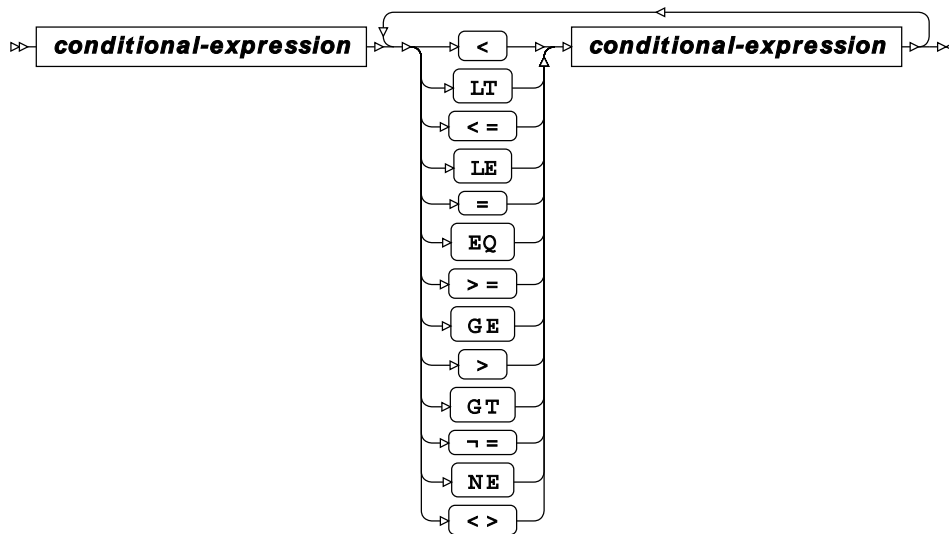
item



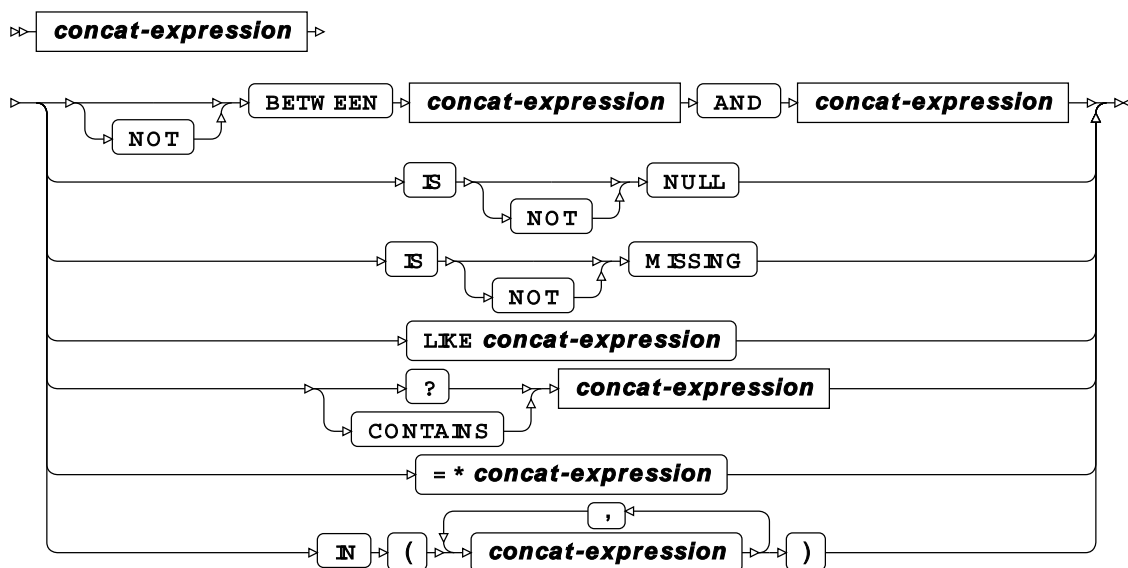
ⁱ See *Column Name* [↗](#) (page 2494).

ⁱⁱ See *SQL expression* [↗](#) (page 2491).

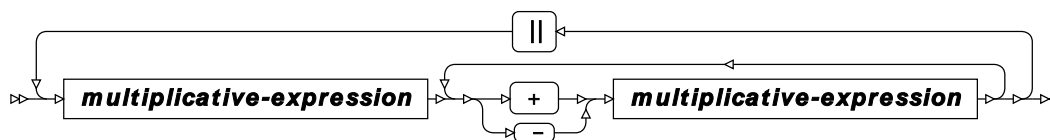
Relational expression



conditional-expression



concat-expression



The diagram illustrates the grammar rule for **primary-expression**. It shows a choice between two paths: one through a box labeled **+** and another through a box labeled **-**. Both paths lead to a box labeled **primary-expression**. From the **primary-expression** box, the path continues to a box labeled **** power-expression**, which then loops back to the input of the **primary-expression** box, indicating a recursive definition for the power operation.

The diagram illustrates the structure of the SQL expression grammar. It shows a sequence of components: a **constant**, a **column name** (marked with a superscript *i*), an **expression-alias**, and a **(SQL expression ⁱⁱ)**. Arrows indicate the flow of the grammar, showing how these components are combined to form the final expression.

ii See *SQL expression* [↗](#) (page 2491).

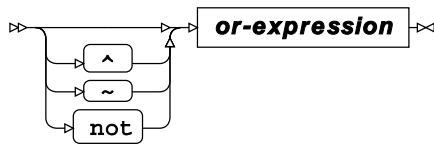
```

graph LR
    i["table expression i"] --> J1(( ))
    J1 --> EXCEPT
    J1 --> INTERSECT
    J1 --> UNION
    J1 --> OUTER_UNION[OUTER UNION]
    EXCEPT --> CORRESPONDING1[CORRESPONDING]
    INTERSECT --> CORRESPONDING1
    UNION --> CORRESPONDING1
    OUTER_UNION --> CORRESPONDING2[CORRESPONDING]
    CORRESPONDING1 --> ALL[ALL]
    CORRESPONDING2 --> ALL
    ALL --> ii["table expression ii"]
    ii --> J1

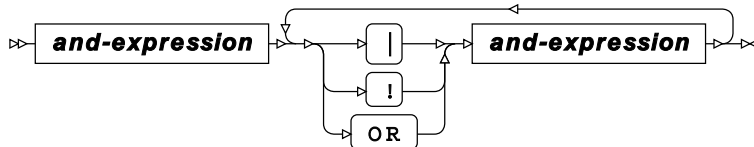
```

ii See *TABLE expression* [↗](#) (page 2495).

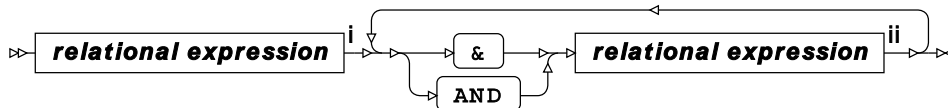
SQL expression



or-expression



and-expression



ⁱ See *Relational expression* [↗](#) (page 2489).

ⁱⁱ See *Relational expression* [↗](#) (page 2489).

Functions

LOWER function



ⁱ See *SQL expression* [↗](#) (page 2491).

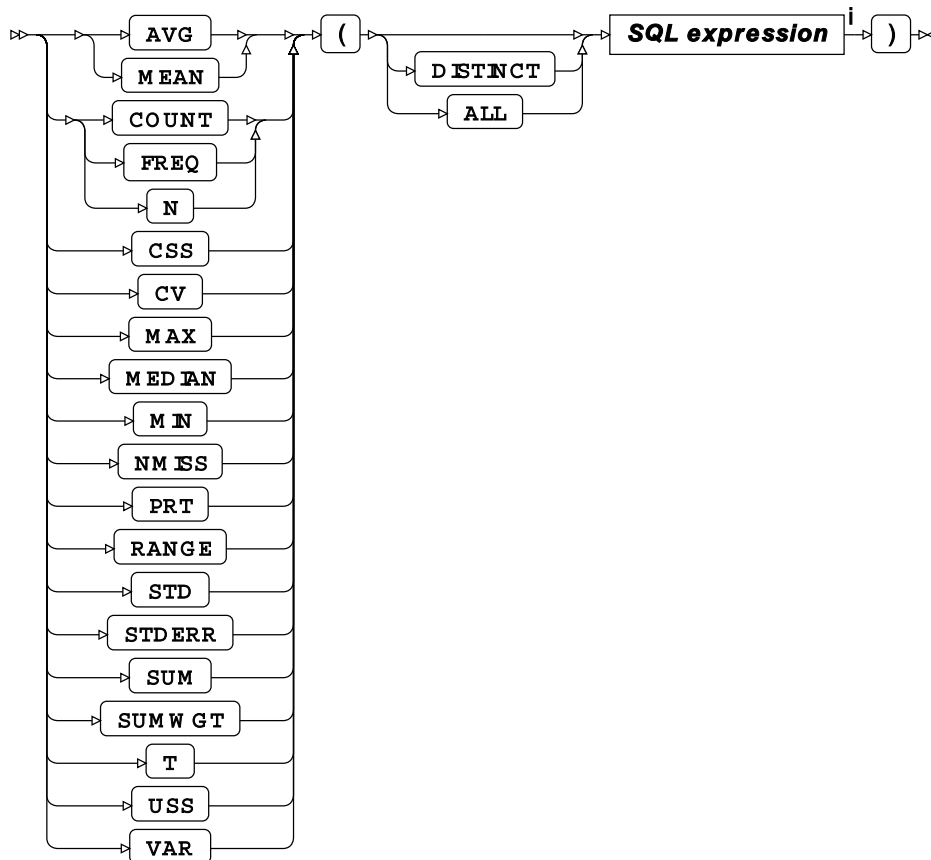
SUBSTRING function



ⁱ See *SQL expression* [↗](#) (page 2491).

ⁱⁱ See *SQL expression* [↗](#) (page 2491).

SUMMARY function



ⁱ See *SQL expression* [↗](#) (page 2491).

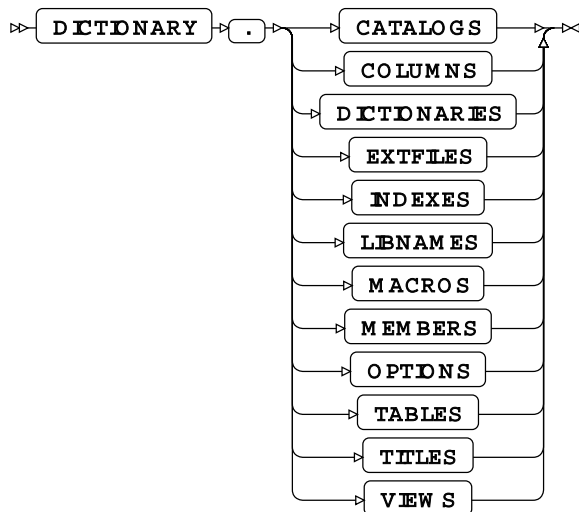
UPPER function



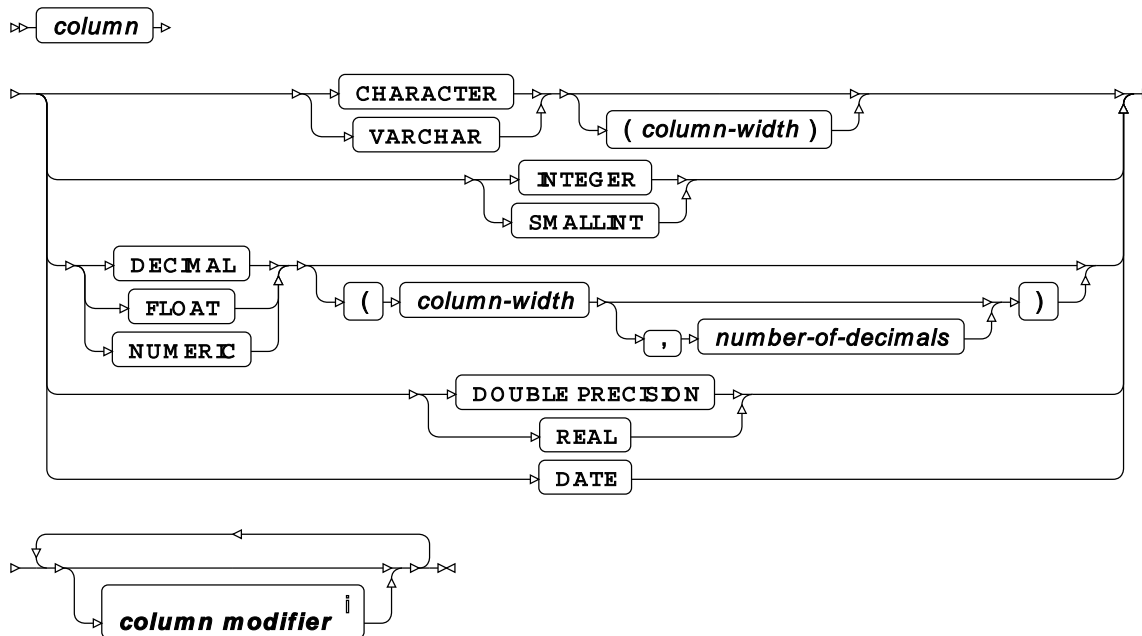
ⁱ See *SQL expression* [↗](#) (page 2491).

Tables

DICTIONARY tables

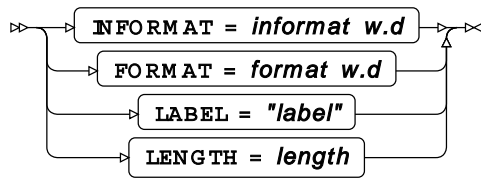


Column Definition

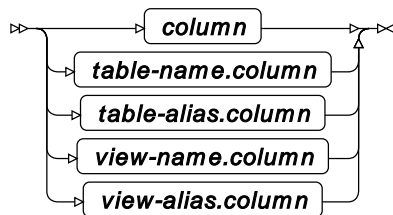


ⁱ See [Column Modifier](#) (page 2494).

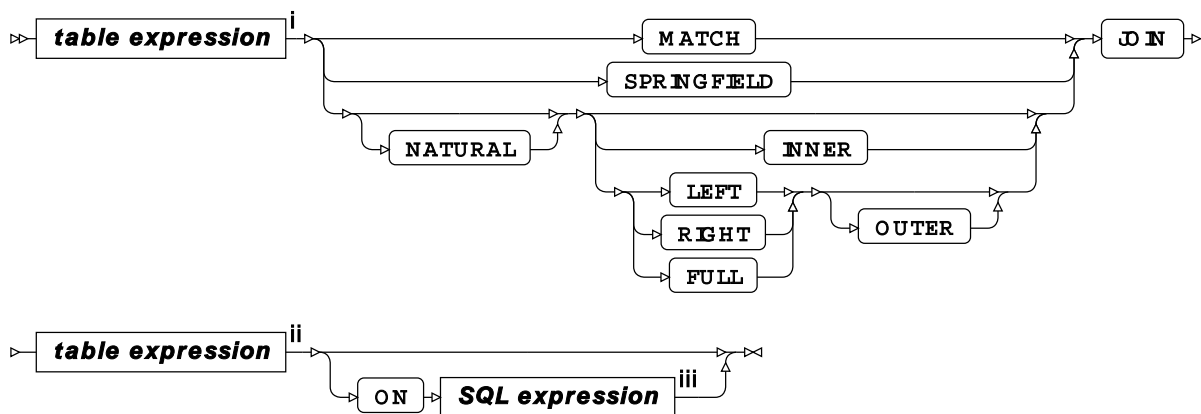
Column Modifier



Column Name



Joined Table

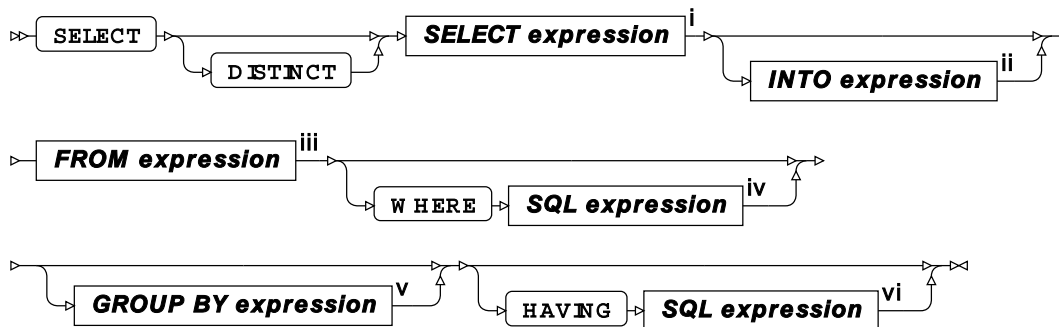


ⁱ See *TABLE expression* [↗](#) (page 2495).

ⁱⁱ See *TABLE expression* [↗](#) (page 2495).

ⁱⁱⁱ See *SQL expression* [↗](#) (page 2491).

TABLE expression



ⁱ See *SELECT expression* [↗](#) (page 2486).

ⁱⁱ See *INTO expression* [↗](#) (page 2487).

ⁱⁱⁱ See *FROM expression* [↗](#) (page 2487).

^{iv} See *SQL expression* [↗](#) (page 2491).

^v See *GROUP BY expression* [↗](#) (page 2488).

^{vi} See *SQL expression* [↗](#) (page 2491).

Connections

CONNECTION-TO



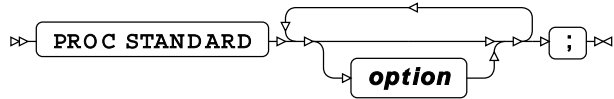
STANDARD procedure

Supported statements

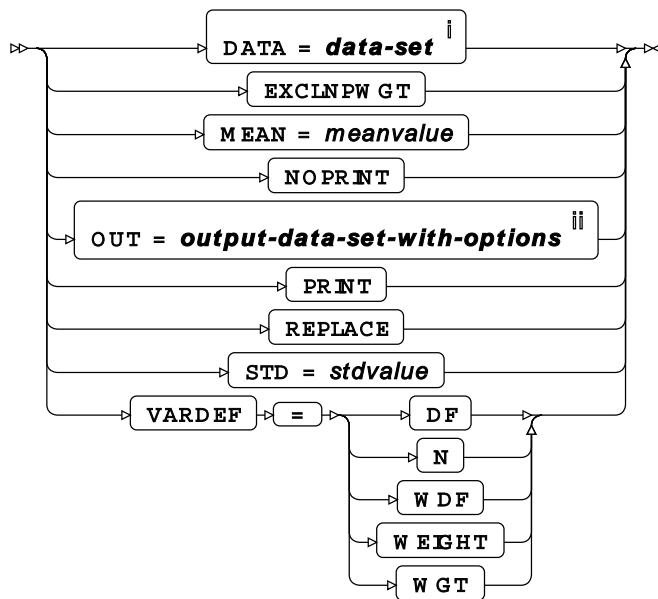
- *PROC STANDARD* [↗](#) (page 2496)
- *BY* [↗](#) (page 2496)
- *FREQ* [↗](#) (page 2497)
- *VAR* [↗](#) (page 2497)
- *WEIGHT* [↗](#) (page 2497)

PROC STANDARD

Standardises data in a dataset to a given mean or standard deviation.



option

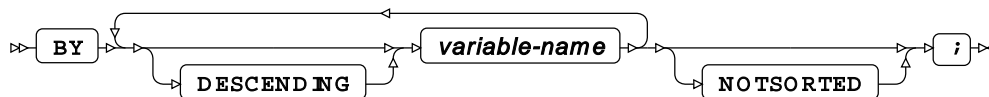


ⁱ See [Input dataset](#) (page 16).

ⁱⁱ See [Output dataset](#) (page 16).

BY

Groups the observations in a dataset using one or more specified variables.



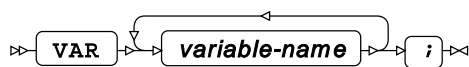
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



VAR

Specifies variables for which to calculate statistics.



WEIGHT

Specifies a variable giving the weight associated with each observation.



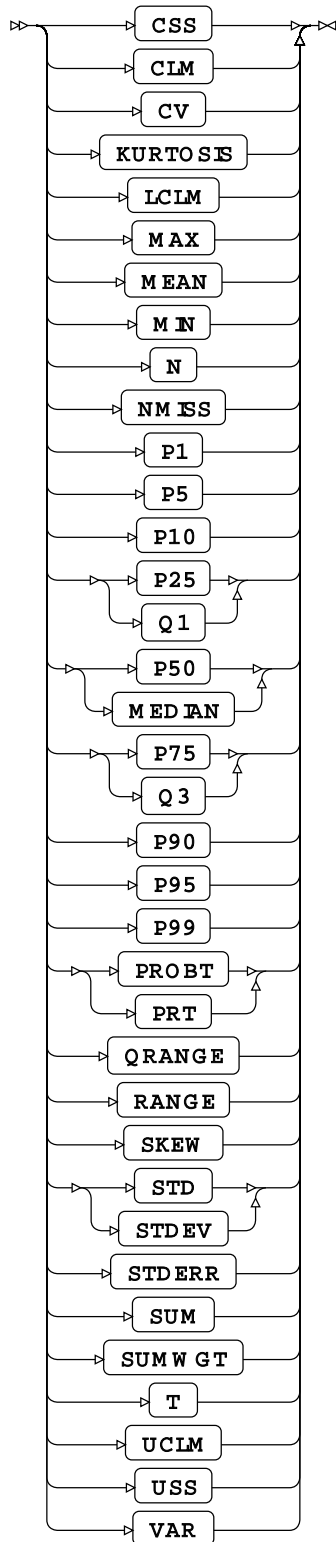
SUMMARY procedure

Supported statements

- *PROC SUMMARY* [↗](#) (page 2499)
- *BY* [↗](#) (page 2501)
- *CLASS* [↗](#) (page 2501)
- *FREQ* [↗](#) (page 2502)
- *ID* [↗](#) (page 2502)
- *OUTPUT* [↗](#) (page 2502)
- *TYPES* [↗](#) (page 2503)
- *VAR* [↗](#) (page 2503)
- *WAYS* [↗](#) (page 2503)
- *WEIGHT* [↗](#) (page 2504)
- *WHERE* [↗](#) (page 2504)

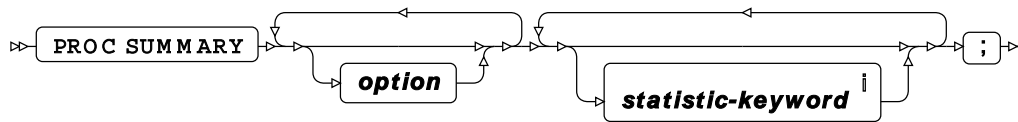
Statistic keywords

The following keywords are used within several statements of this procedure.



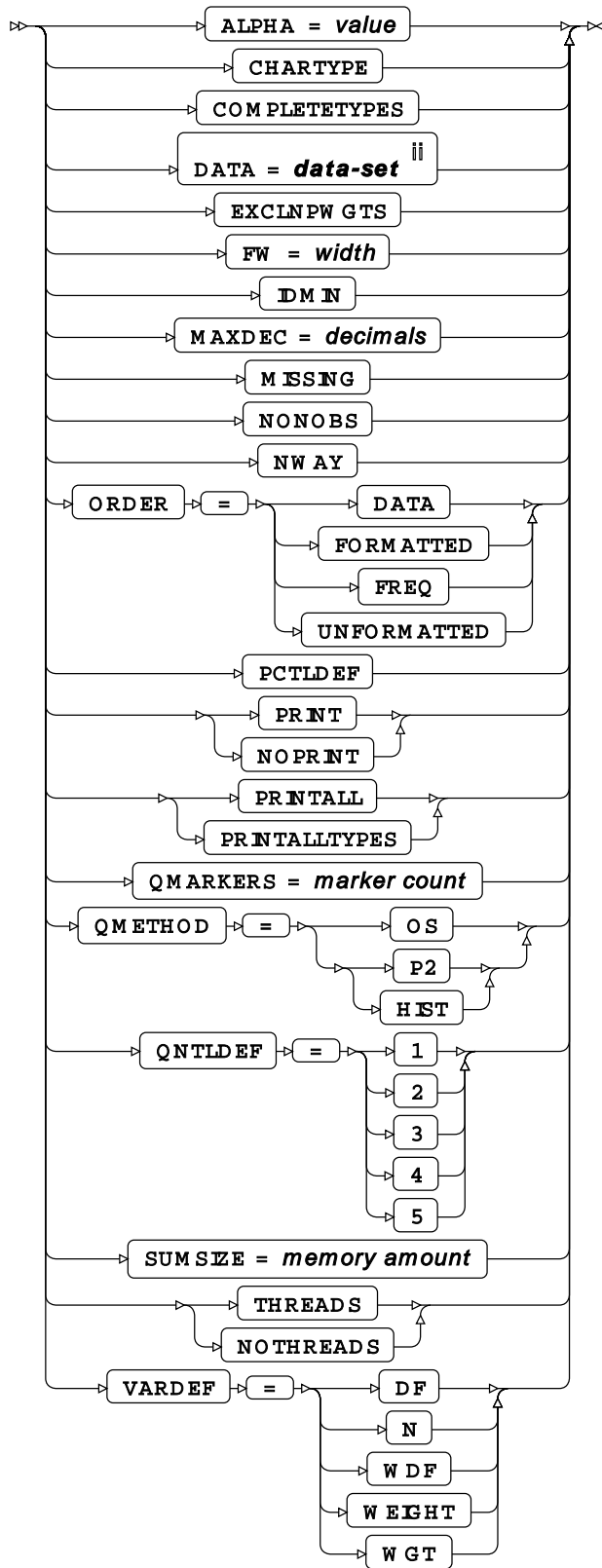
PROC SUMMARY

Calculates elementary statistics for a dataset.



ⁱ See *Statistic keywords* [↗](#) (page 2498).

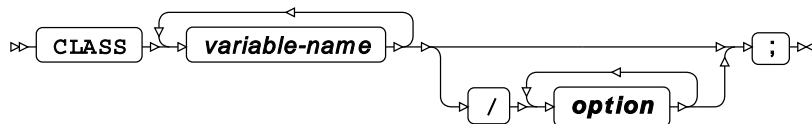
option



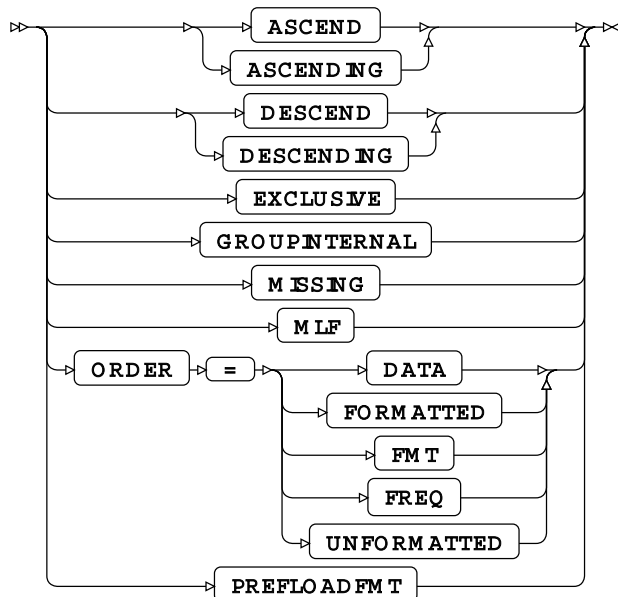
ii See *Input dataset* [↗](#) (page 16).

CLASS

Specifies variables (within a BY group), by which observations are to be grouped.

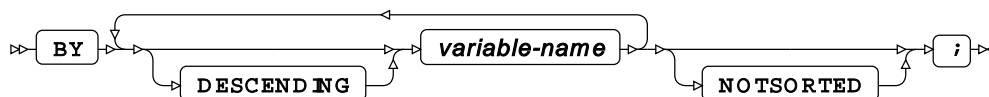


option



BY

Groups the observations in a dataset using one or more specified variables.



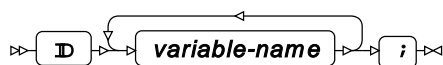
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



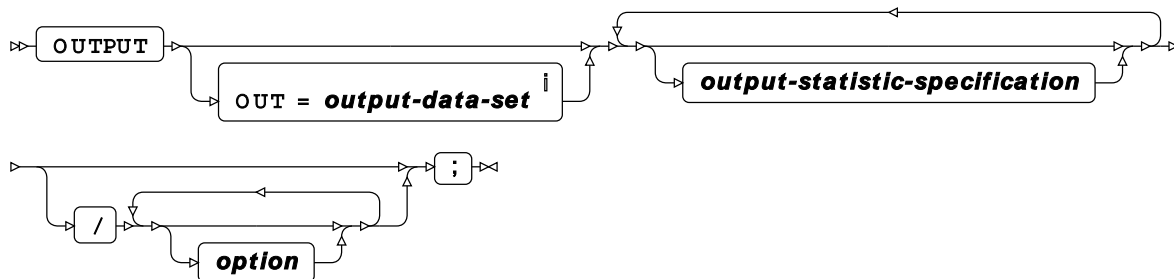
ID

Identifies the relevant observations in the output by using one or more specified variable names.



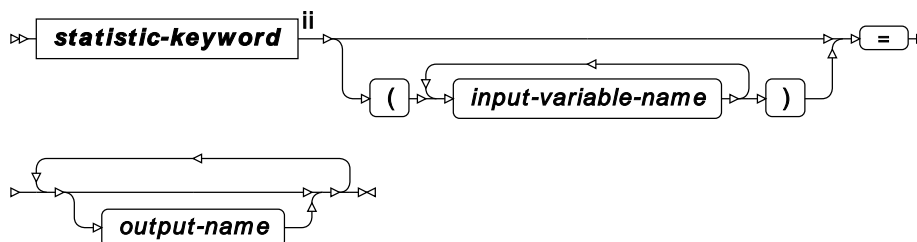
OUTPUT

Creates an output dataset containing data given by one or more statistic keyword specifications.



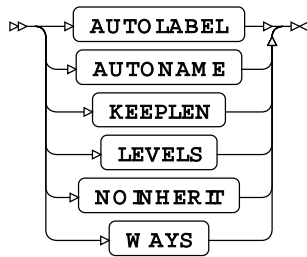
ⁱ See *Output dataset* [↗](#) (page 16).

output-statistic-specification



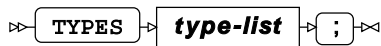
ⁱⁱ See *Statistic keywords* [↗](#) (page 2498).

option

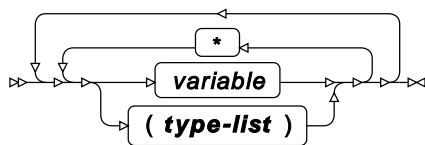


TYPES

Restricts output to subsets of `CLASS` variables.

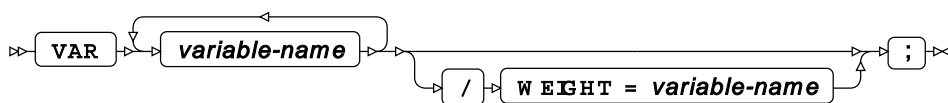


type-list



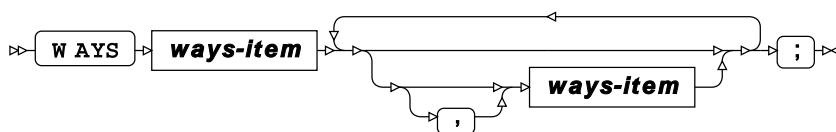
VAR

Specifies variables for which to calculate statistics.

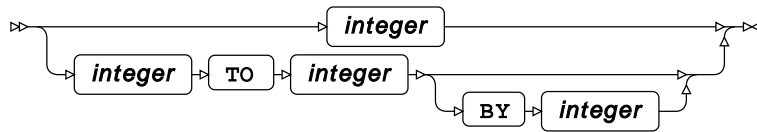


WAYS

Restricts outputs to the numbers of `WAYS` given. Examples can be a one dimensional table or a two dimensional table output, or both.

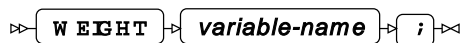


ways-item



WEIGHT

Specifies a variable giving the weight associated with each observation.



WHERE

Restricts the observations to be processed.

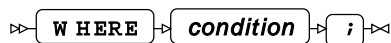


TABLEAU procedure

Supported statements

- *PROC TABLEAU* [↗](#) (page 2504)
- *UPLOAD* [↗](#) (page 2505)

PROC TABLEAU

Exports a dataset to a Tableau Data Extract file.

To use the `TABLEAU` procedure, you must have the Tableau SDK installed.

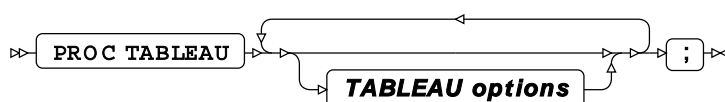
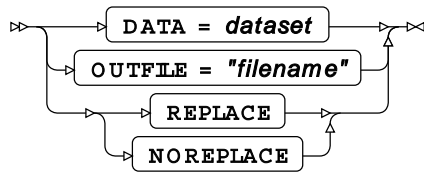
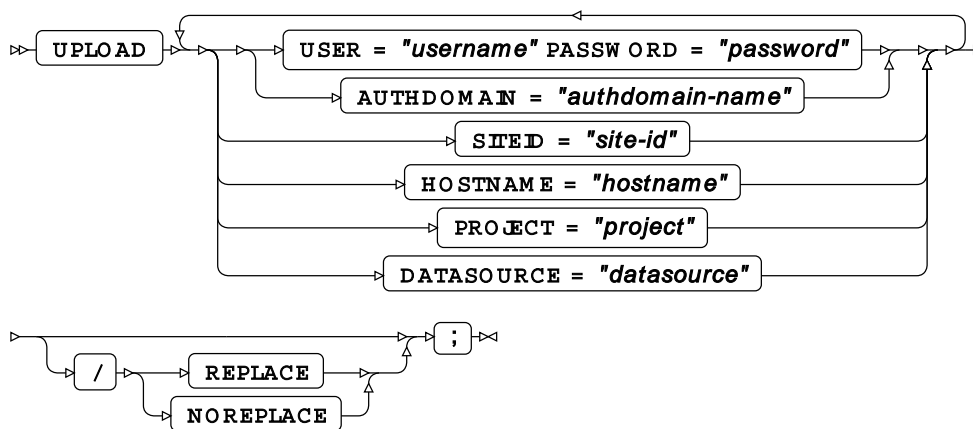


TABLEAU options



UPLOAD

Specifies the remote host to which the dataset is exported.



TABULATE procedure

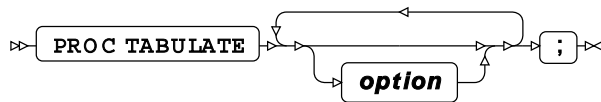
Supported statements

- *PROC TABULATE* [↗](#) (page 2506)
- *BY* [↗](#) (page 2508)
- *CLASS* [↗](#) (page 2509)
- *CLASSLEV* [↗](#) (page 2509)
- *FREQ* [↗](#) (page 2509)
- *KEYLABEL* [↗](#) (page 2510)
- *KEYWORD* [↗](#) (page 2510)
- *TABLE* [↗](#) (page 2510)
- *VAR* [↗](#) (page 2511)
- *WEIGHT* [↗](#) (page 2511)

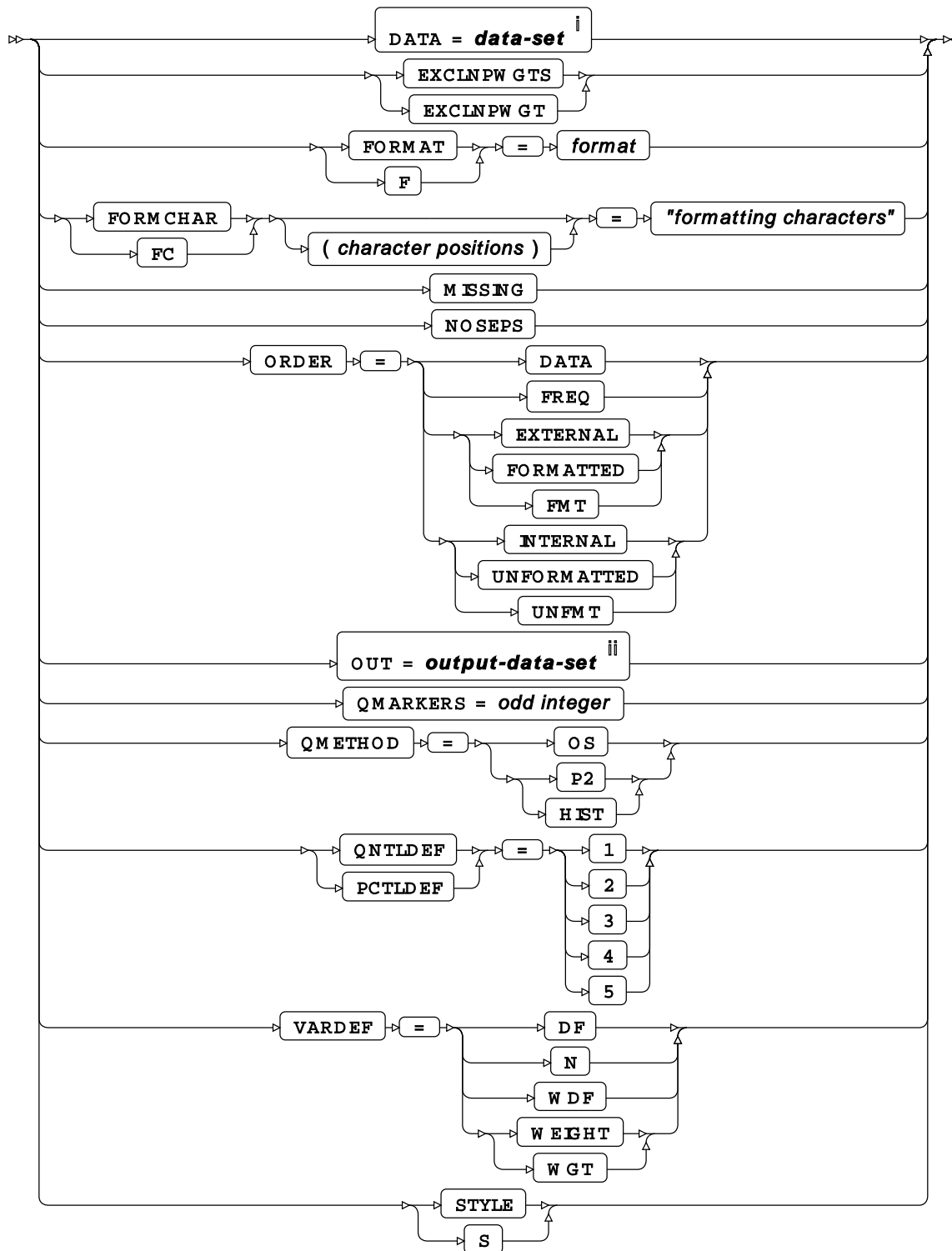
- WHERE [↗](#) (page 2511)

PROC TABULATE

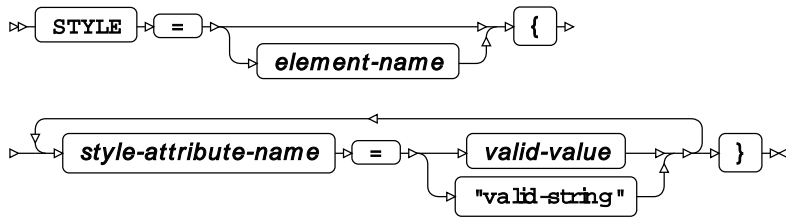
Creates tabulated summaries of a dataset.



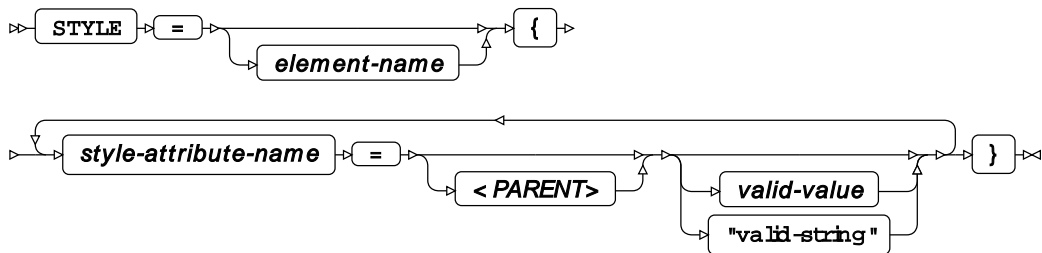
option

ⁱ See [Input dataset](#) (page 16).ⁱⁱ See [Output dataset](#) (page 16).

style - when used at the PROC TABULATE level

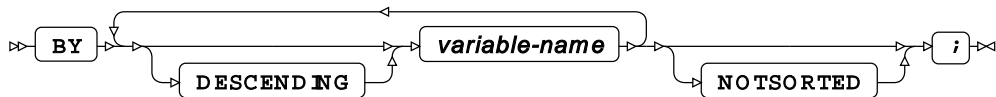


style - when used in a PROC TABULATE option



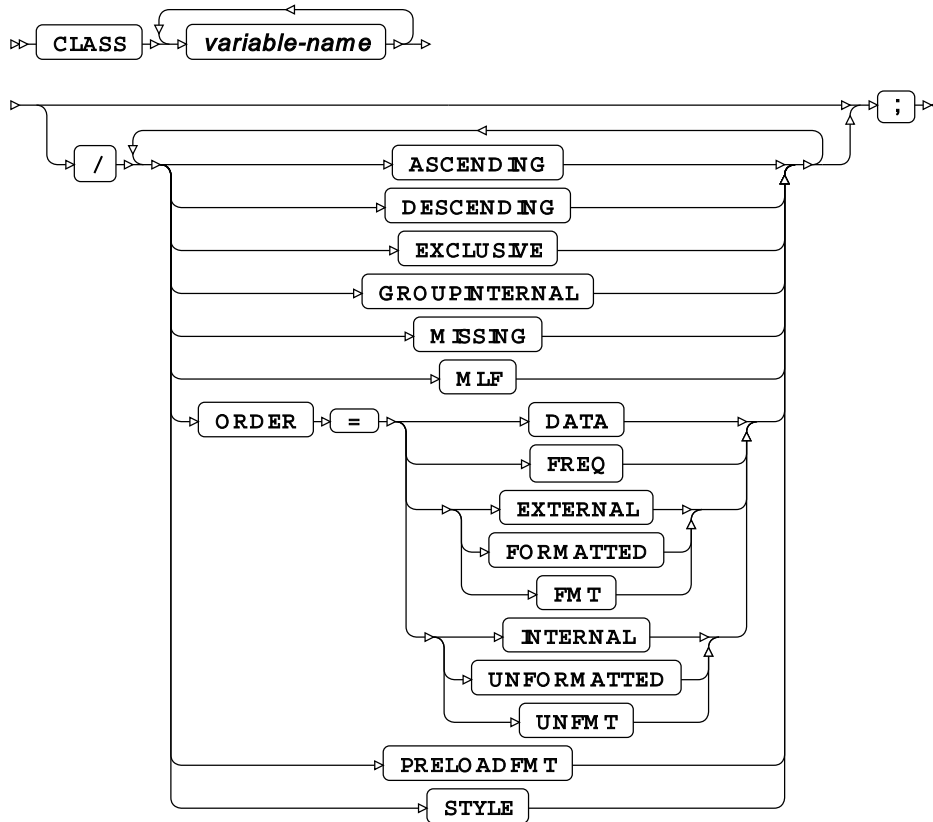
BY

Groups the observations in a dataset using one or more specified variables.



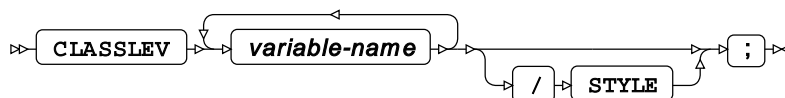
CLASS

Specifies variables by which data in the output table are categorised.



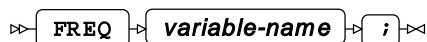
CLASSLEV

Specifies a style to be associated with a class variable.



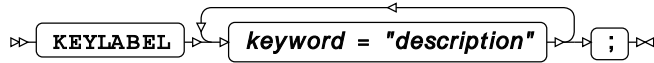
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.



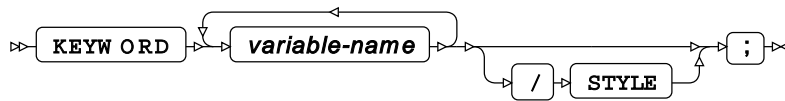
KEYLABEL

Uses this label in place of the `KEYWORD` in the output table.



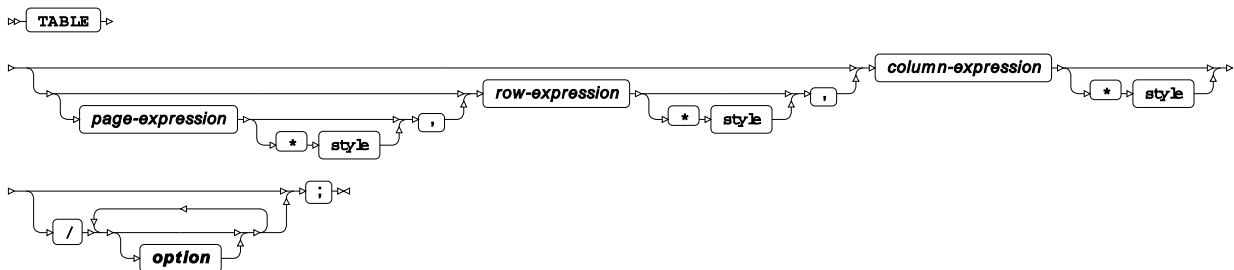
KEYWORD

Specifies a style to be associated with a `KEYWORD`.

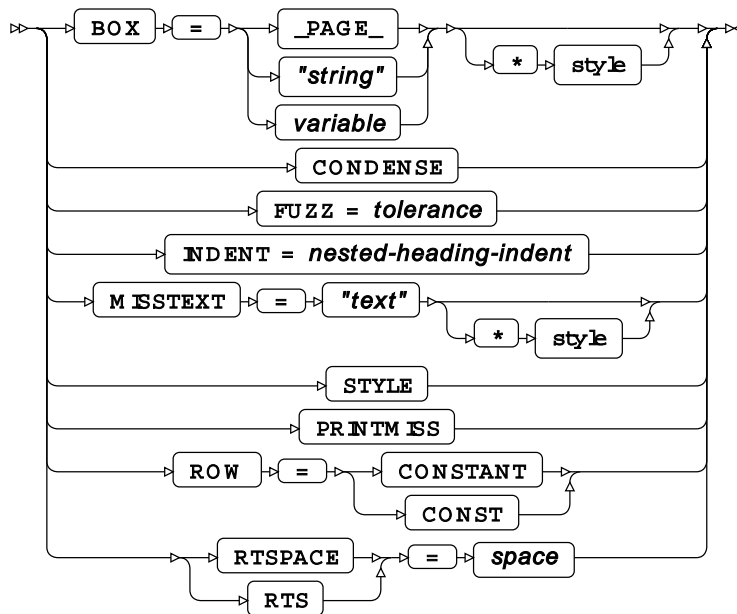


TABLE

Defines the classes, variables and statistics that will form the rows and columns of an output table.

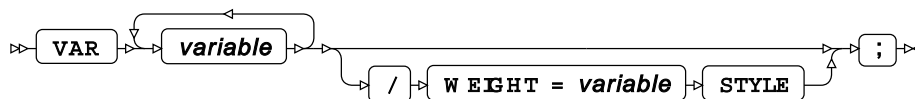


option



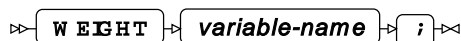
VAR

Specifies analysis variables to be used in the table.



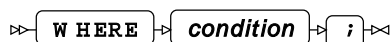
WEIGHT

Specifies a variable giving the weight associated with each observation.



WHERE

Restricts the observations to be processed.



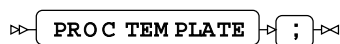
TEMPLATE procedure

Supported statements

- *PROC TEMPLATE* [↗](#) (page 2512)
- *DELETE* [↗](#) (page 2512)
- *EDIT* [↗](#) (page 2513)
- *LINK* [↗](#) (page 2513)
- *LIST* [↗](#) (page 2513)
- *PATH* [↗](#) (page 2515)
- *SOURCE* [↗](#) (page 2516)
- *DEFINE COLUMN* [↗](#) (page 2517)
- *DEFINE FOOTER* [↗](#) (page 2518)
- *DEFINE HEADER* [↗](#) (page 2518)
- *DEFINE STYLE* [↗](#) (page 2518)
- *DEFINE TABLE* [↗](#) (page 2519)
- *DEFINE TAGSET* [↗](#) (page 2520)

PROC TEMPLATE

Parses table definitions and store them in an item store for later retrieval.



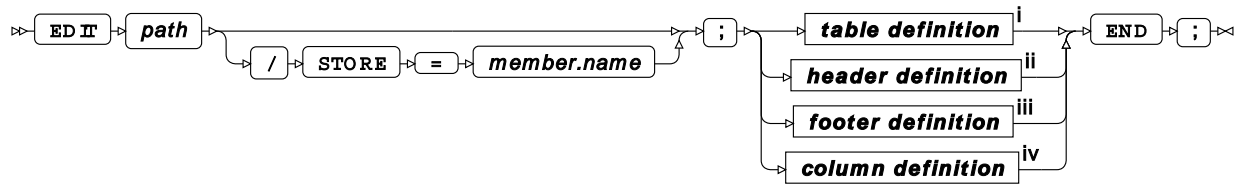
DELETE

Removes a template item, or removes template from a stored device.



EDIT

Redefines an existing table, header, footer or column.



ⁱ See *TABLE definition* [↗](#) (page 2525).

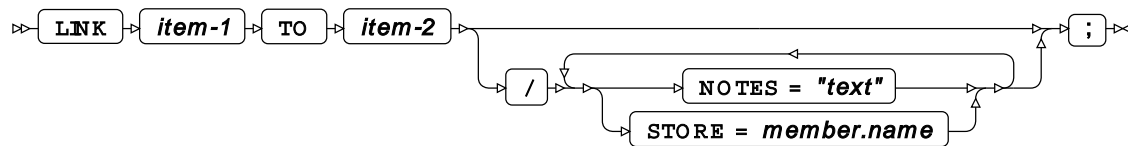
ⁱⁱ See *HEADER definition* [↗](#) (page 2527).

ⁱⁱⁱ See *FOOTER definition* [↗](#) (page 2529).

^{iv} See *COLUMN definition* [↗](#) (page 2531).

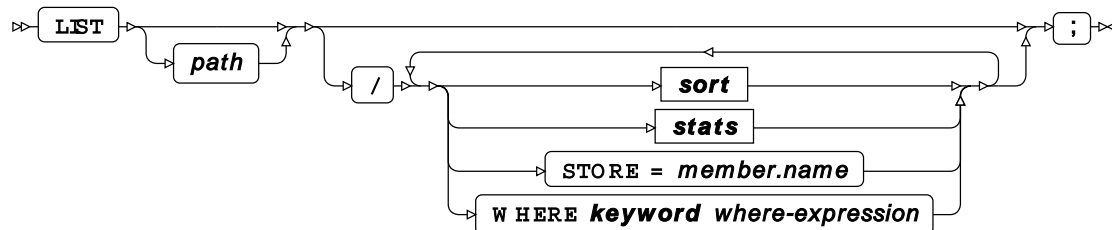
LINK

Links the first item to the second, adding optional parameters.

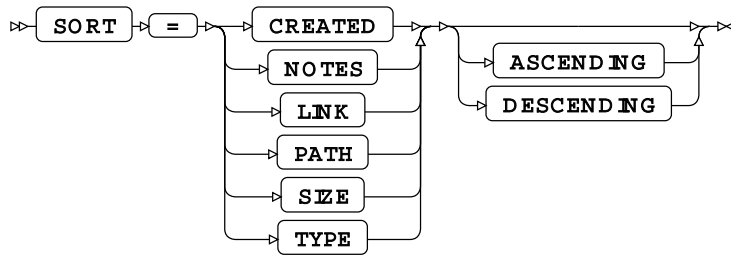


LIST

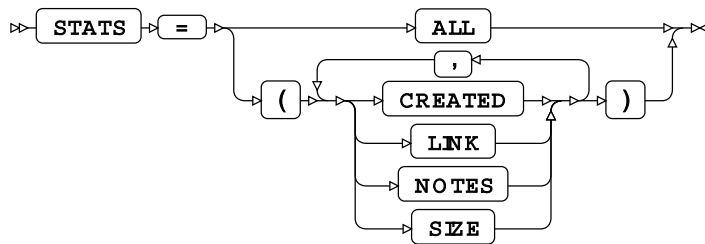
Lists the templates.



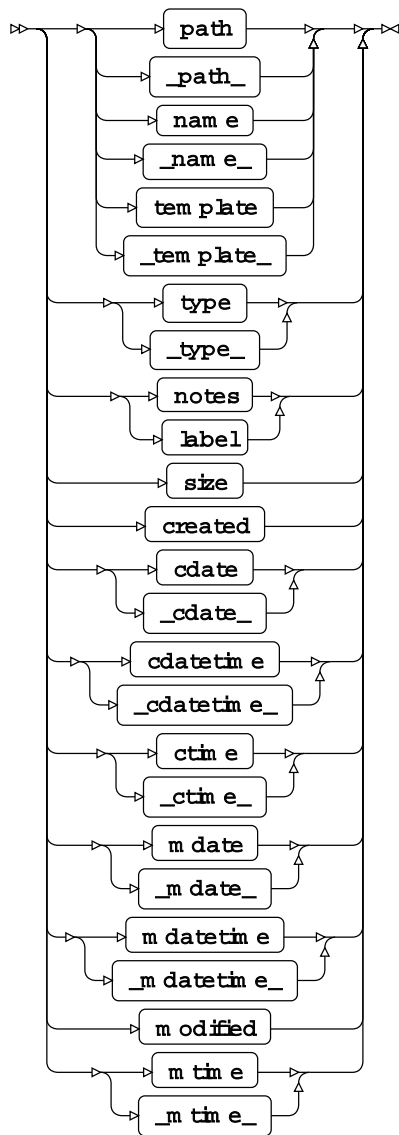
sort



stats

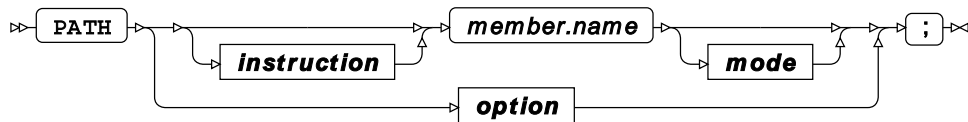


keyword

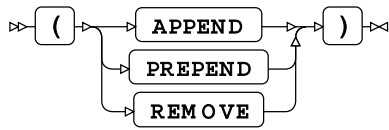


PATH

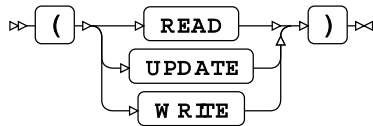
Defines a path to locate a template in an itemstore.



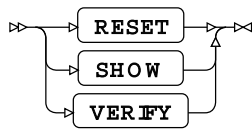
instruction



mode

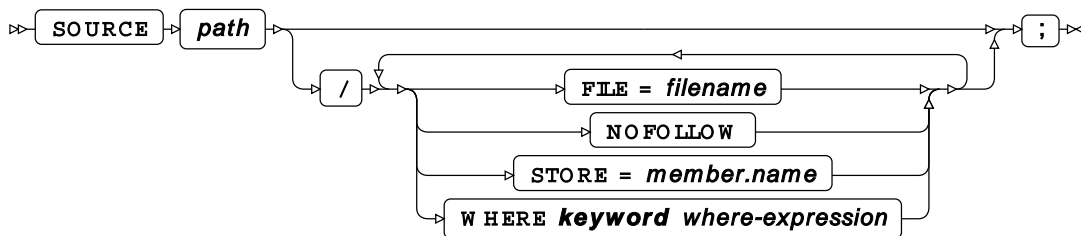


option

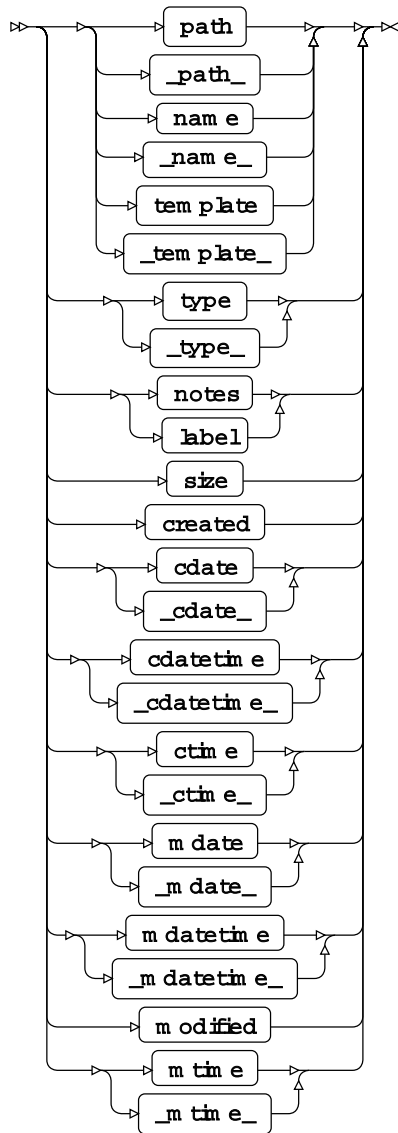


SOURCE

Generates a `TEMPLATE` procedure statement from a stored template.

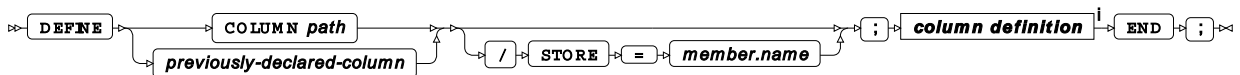


keyword



DEFINE COLUMN

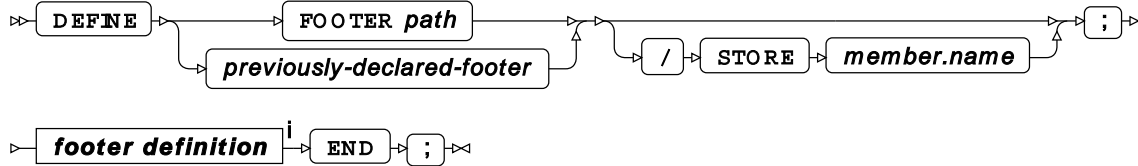
Specifies the appearance of a column.



ⁱ See [COLUMN definition](#) (page 2531).

DEFINE FOOTER

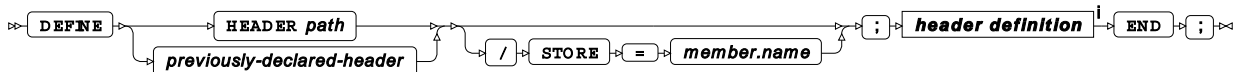
Specifies the appearance of a table footer.



ⁱ See *FOOTER definition* [↗](#) (page 2529).

DEFINE HEADER

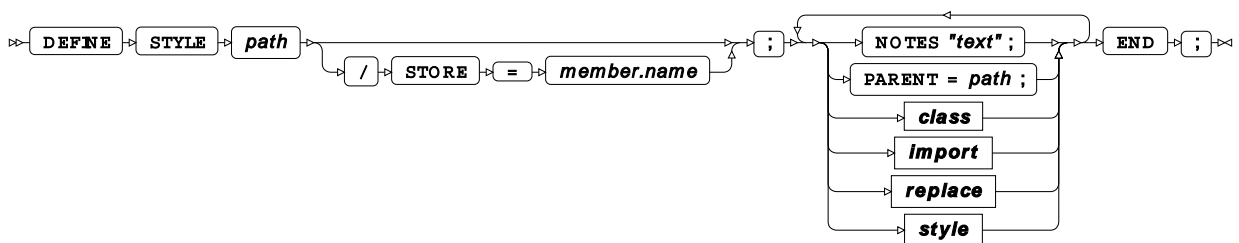
Specifies the appearance of table or column headers.



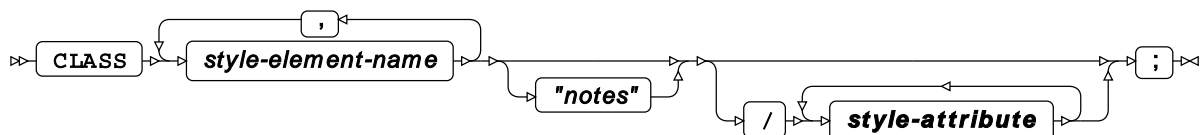
ⁱ See *HEADER definition* [↗](#) (page 2527).

DEFINE STYLE

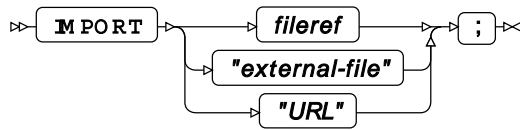
Creates a new style template. Enables the classification, replacement, and adaption of new styles from previous sources.



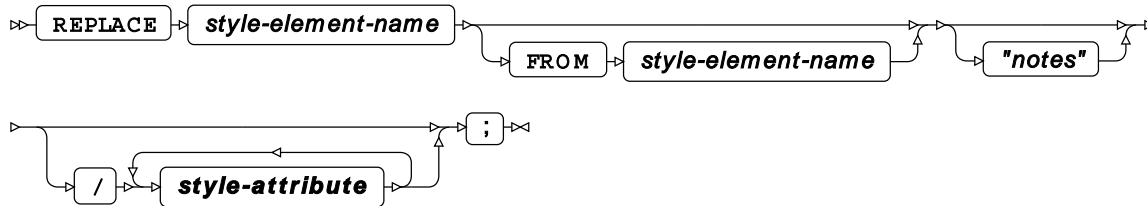
class



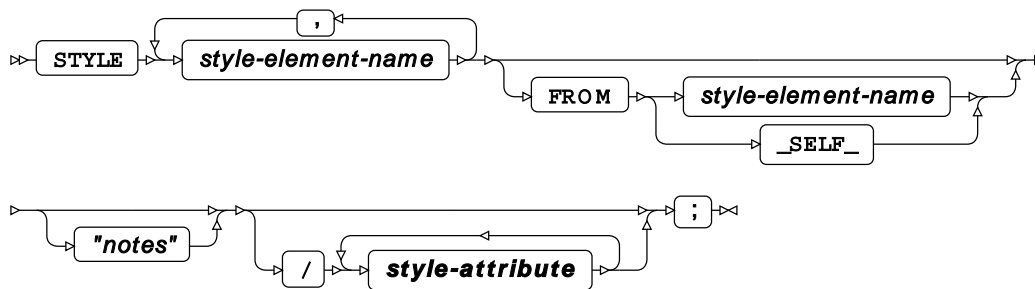
import



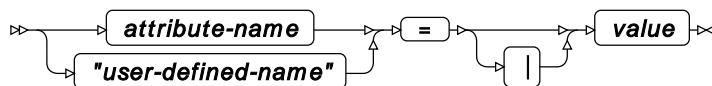
replace



style



style-attribute



DEFINE TABLE

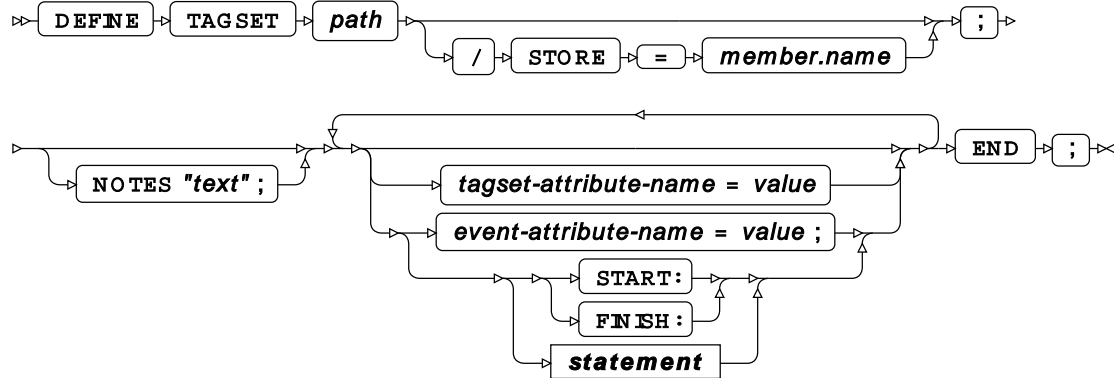
Specifies the appearance of a table.



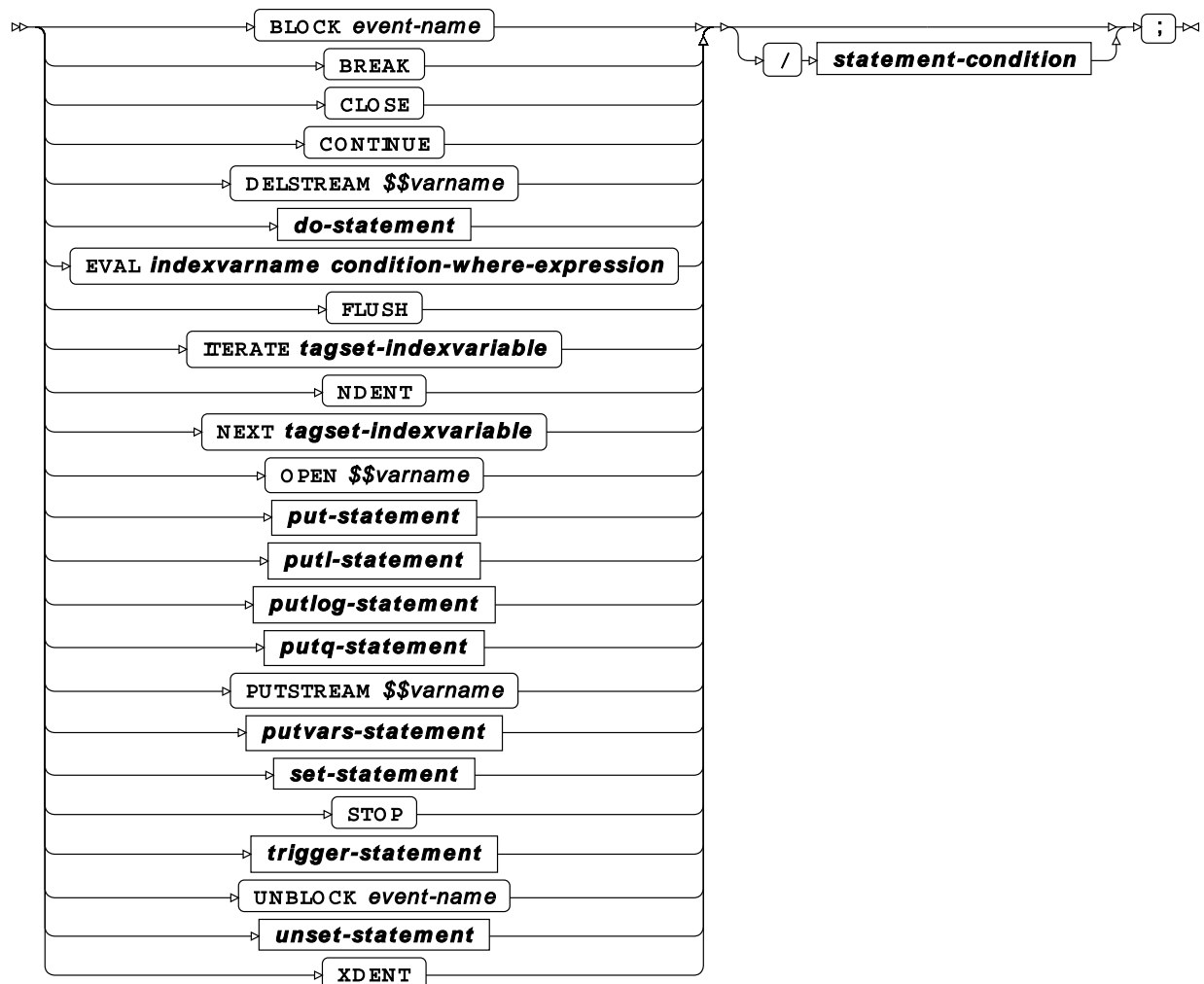
ⁱ See [TABLE definition](#) (page 2525).

DEFINE TAGSET

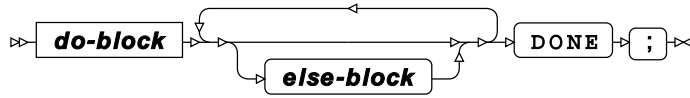
Defines tables, headers, footers and columns.



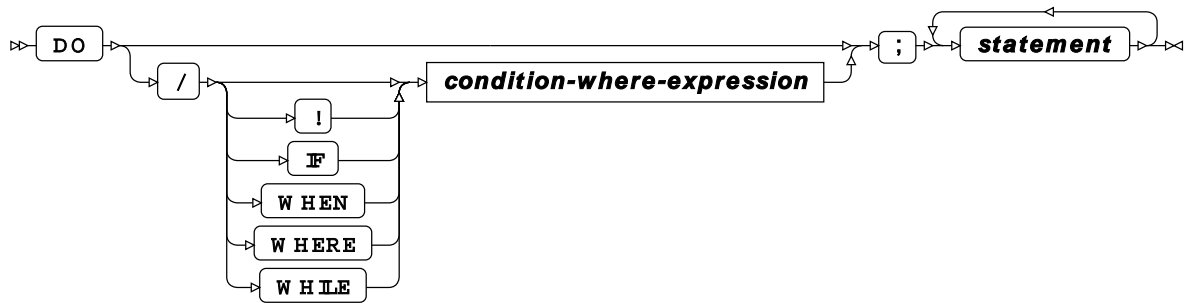
statement



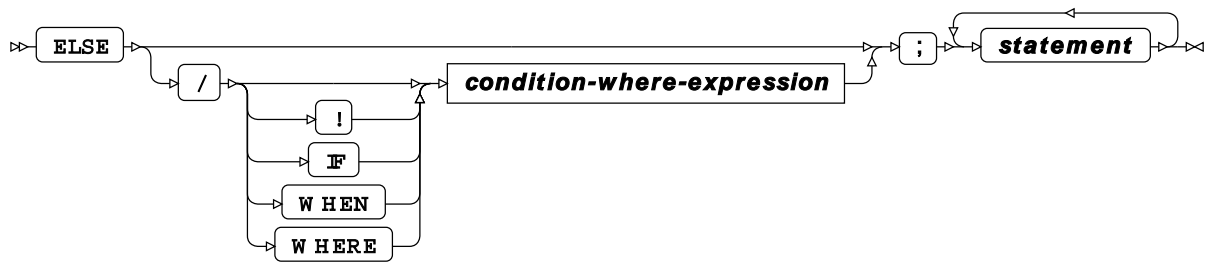
do-statement



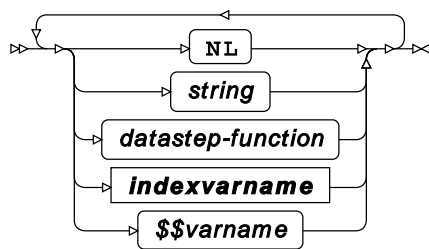
do-block



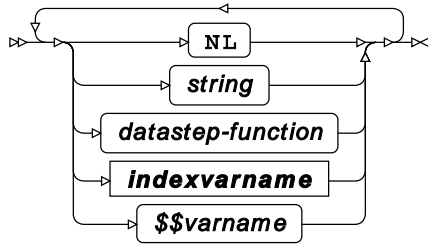
else-block



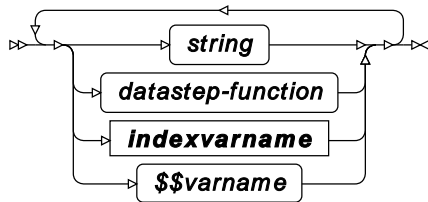
put-statement



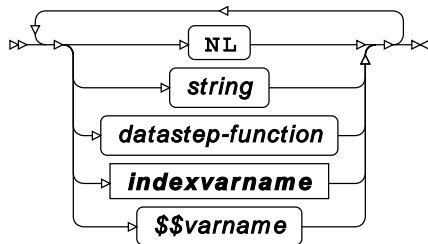
putl-statement



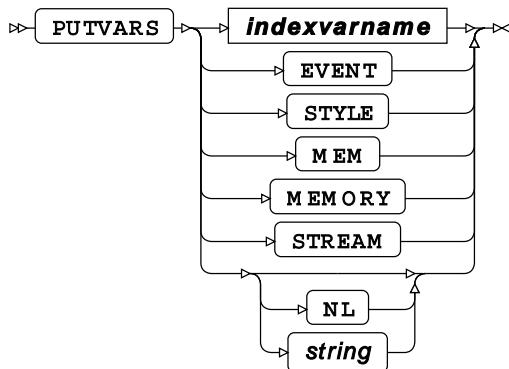
putlog-statement



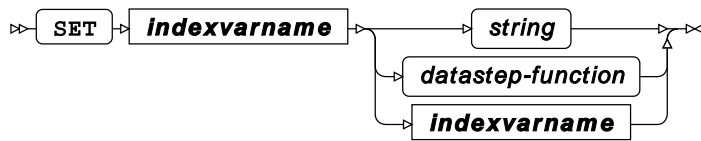
putq-statement



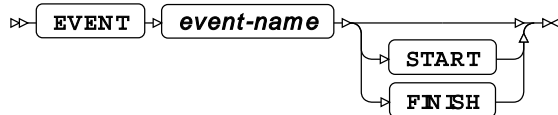
putvars-statement



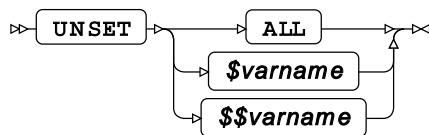
set-statement



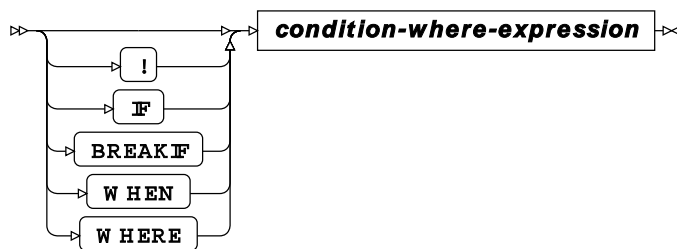
trigger-statement



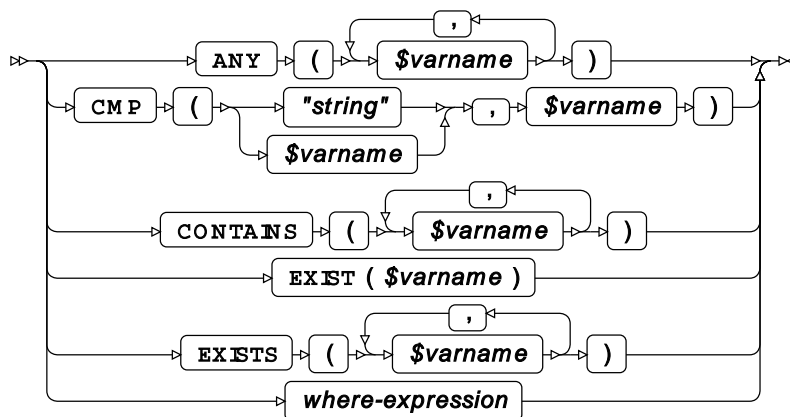
unset-statement



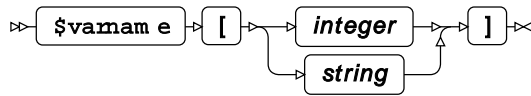
statement-condition



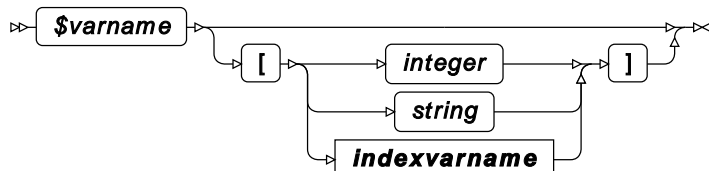
condition-where-expression



tagset-indexvariable



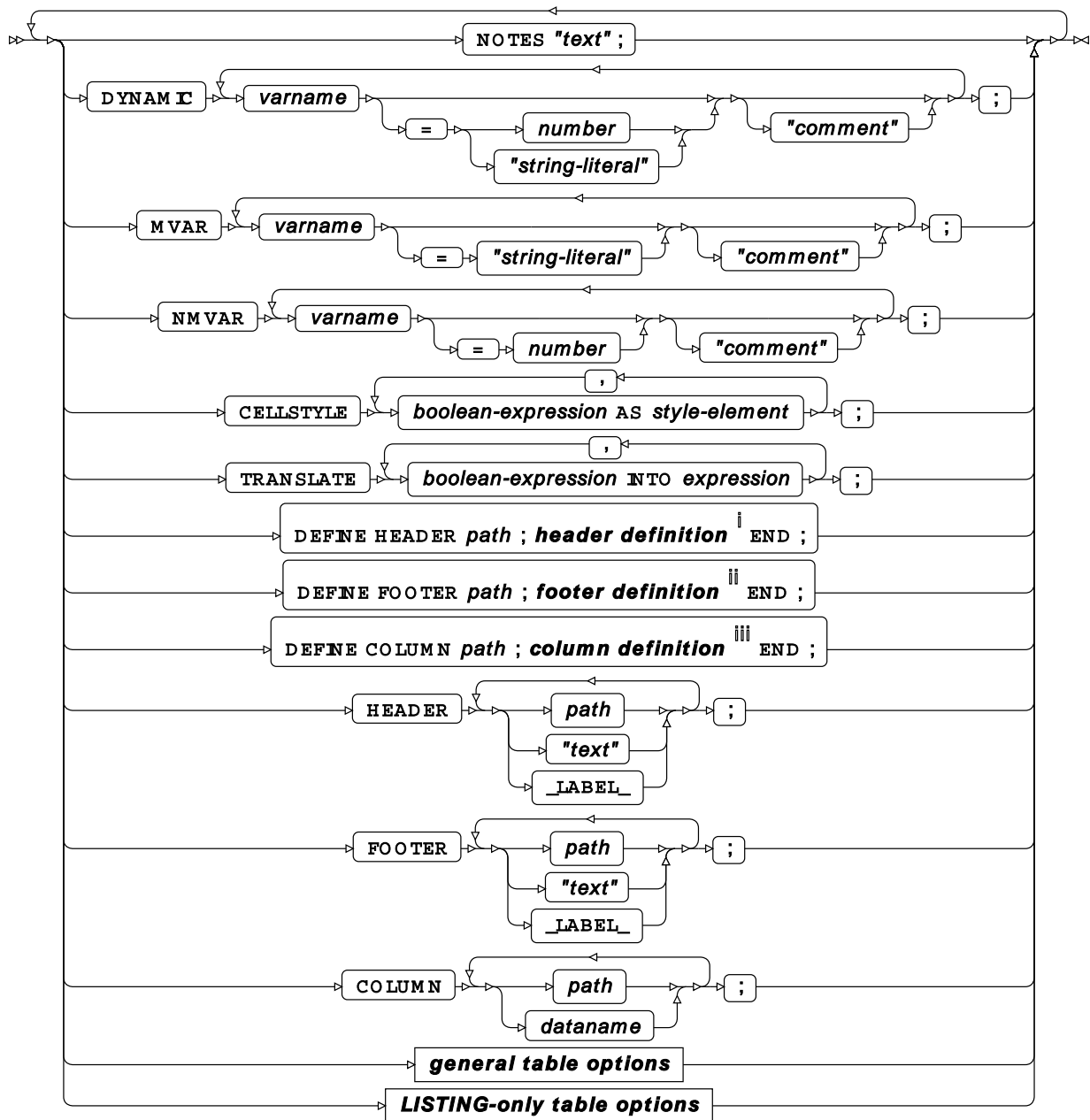
indexvarname



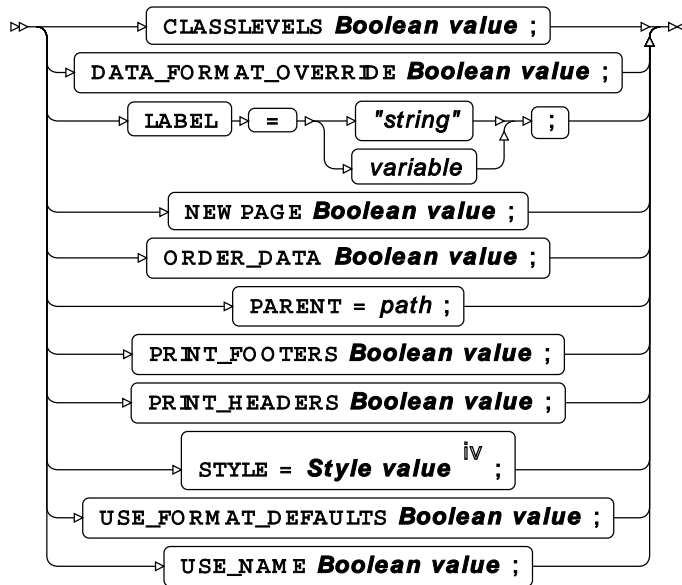
Components

- *TABLE* definition [↗](#) (page 2525)
- *HEADER* definition [↗](#) (page 2527)
- *FOOTER* definition [↗](#) (page 2529)
- *COLUMN* definition [↗](#) (page 2531)
- *STYLE* [↗](#) (page 2534)

TABLE definition

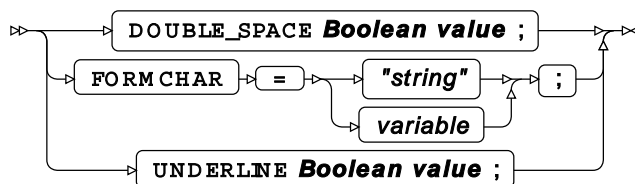
ⁱ See *HEADER* definition [\[link\]](#) (page 2527).ⁱⁱ See *FOOTER* definition [\[link\]](#) (page 2529).ⁱⁱⁱ See *COLUMN* definition [\[link\]](#) (page 2531).

general table options

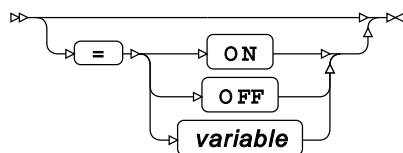


^{iv} See [STYLE](#) (page 2534).

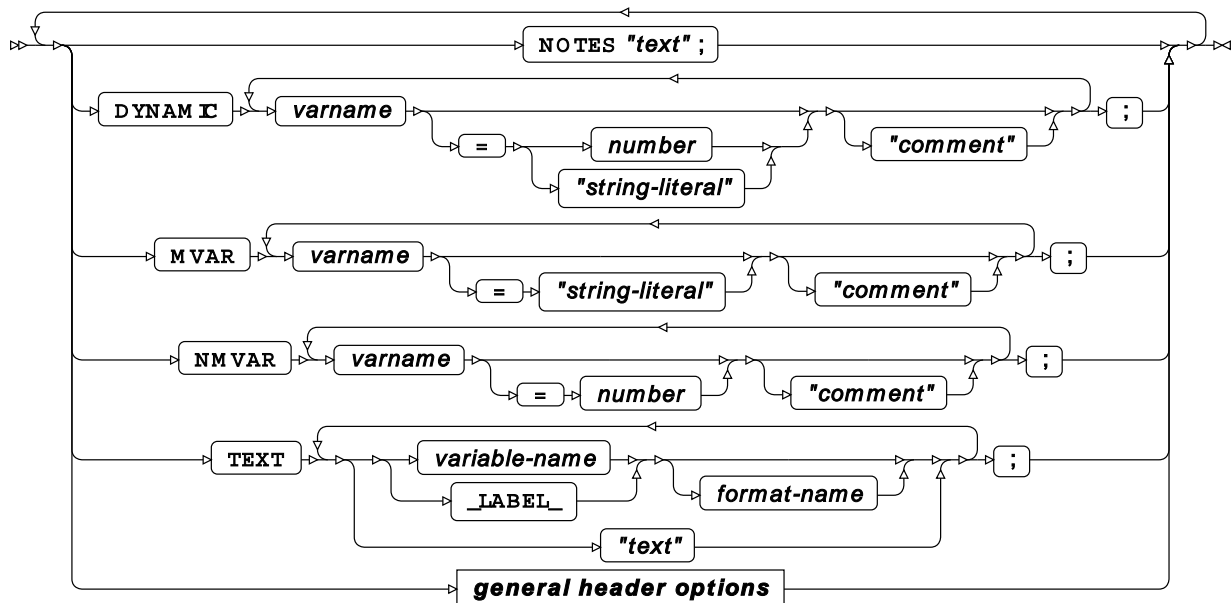
LISTING-only table options



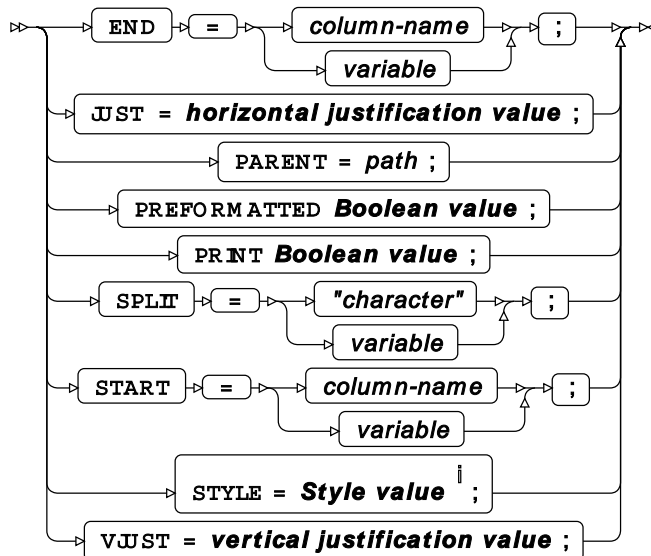
Boolean value



HEADER definition

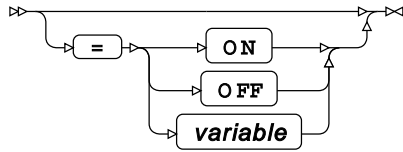


general header options

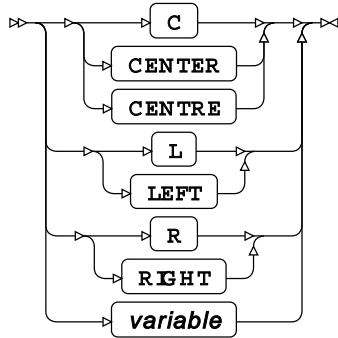


ⁱ See [STYLE](#) (page 2534).

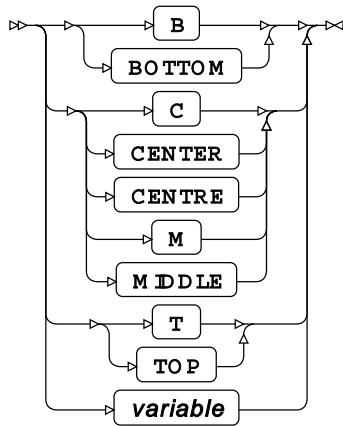
Boolean value



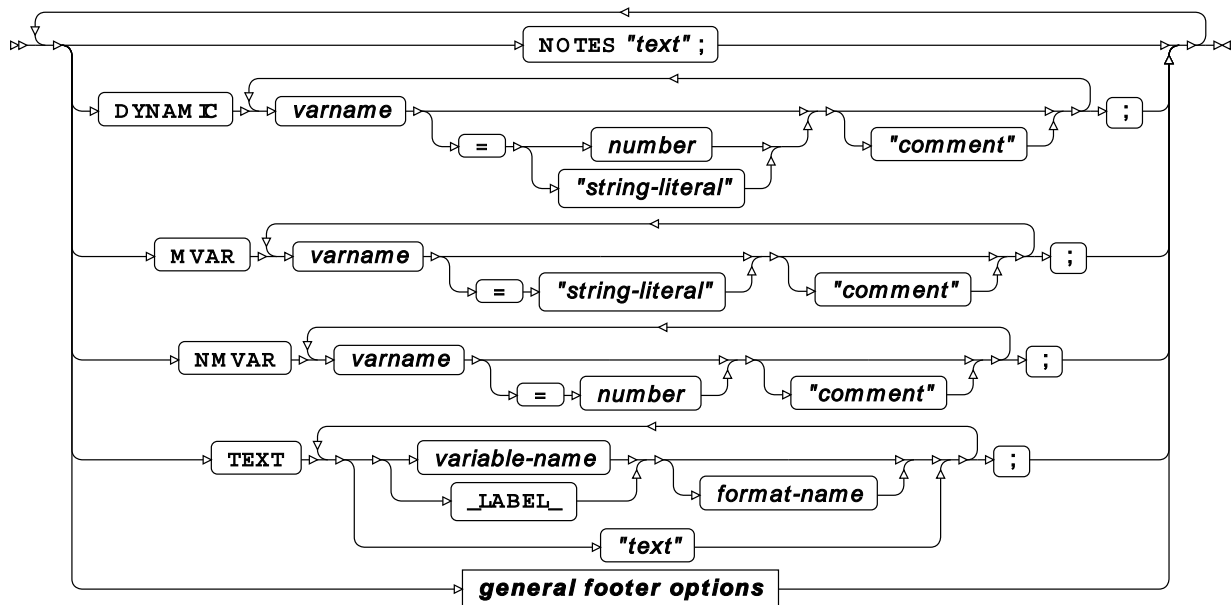
horizontal justification value



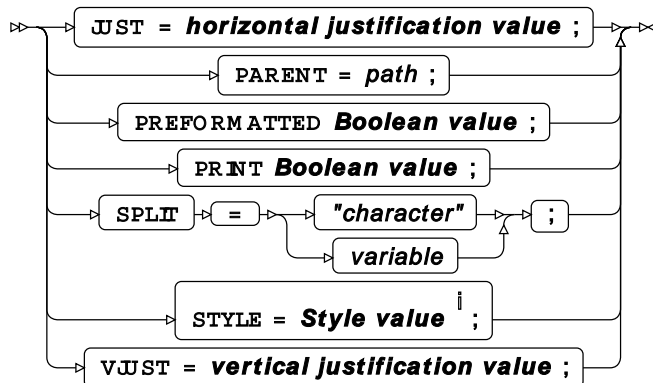
vertical justification value



FOOTER definition

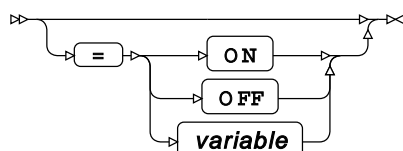


general footer options

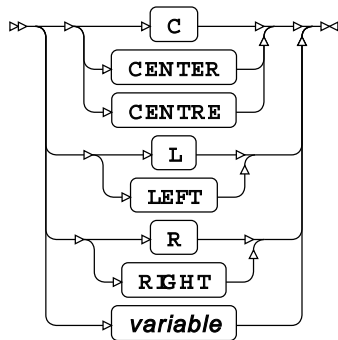


ⁱ See [STYLE](#) (page 2534).

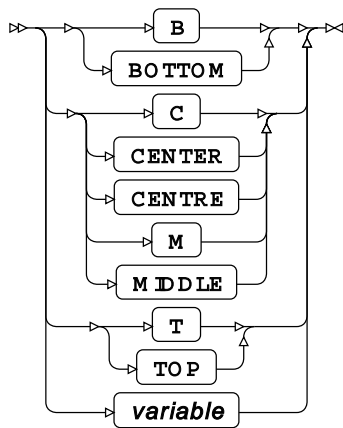
Boolean value



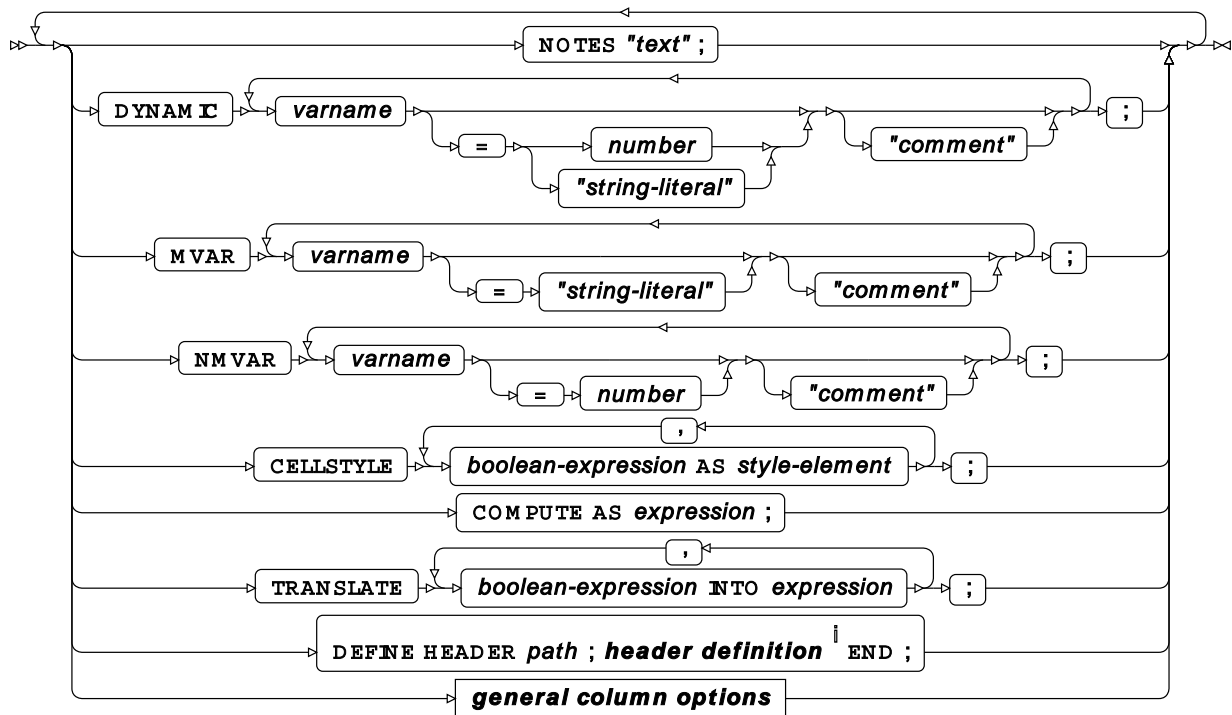
horizontal justification value



vertical justification value

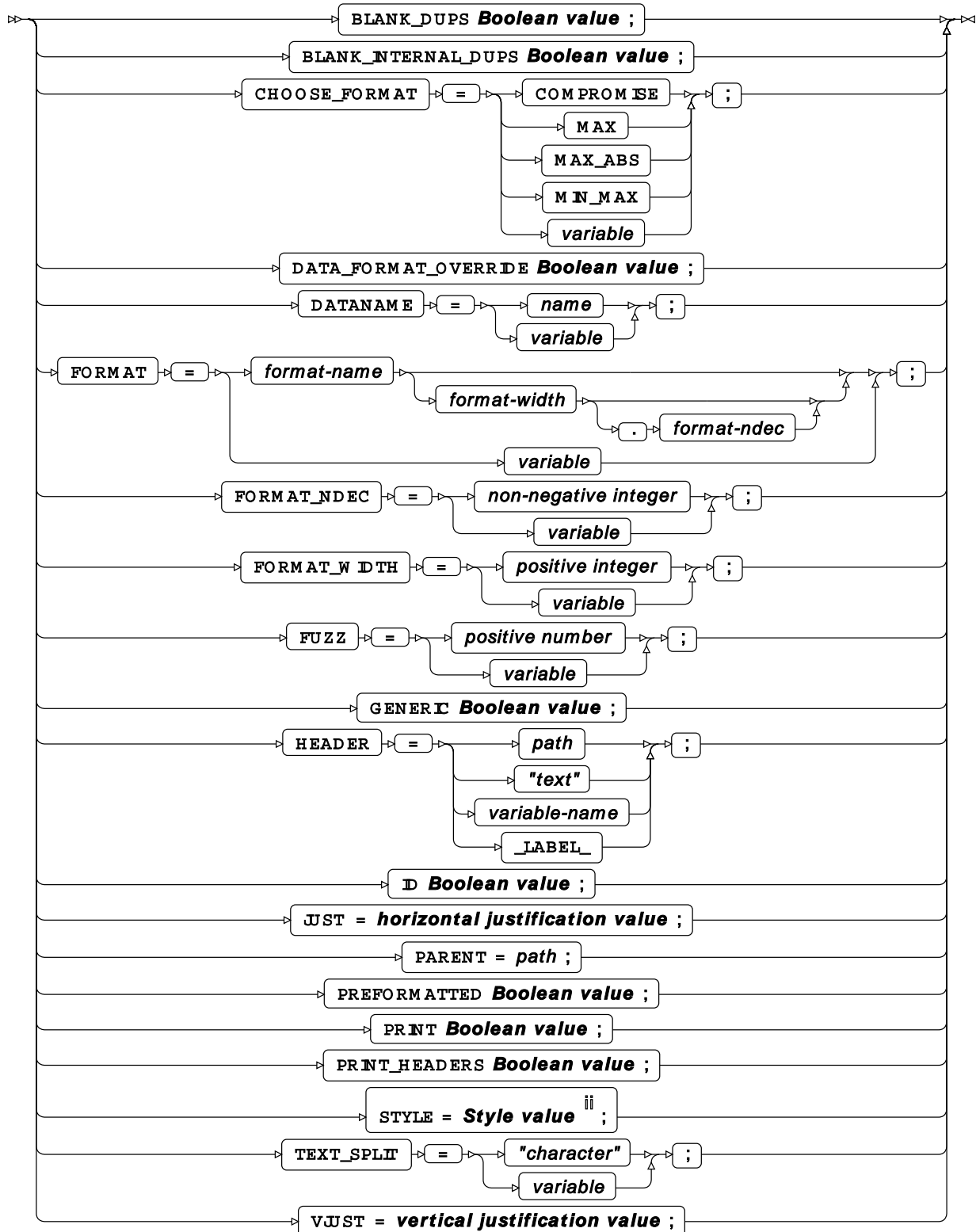


COLUMN definition



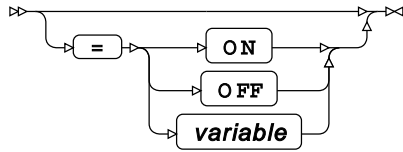
ⁱ See *HEADER definition* [↗](#) (page 2527).

general column options

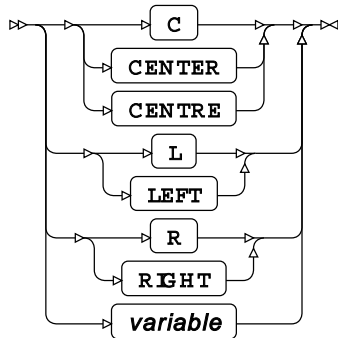


ⁱⁱ See [STYLE](#) (page 2534).

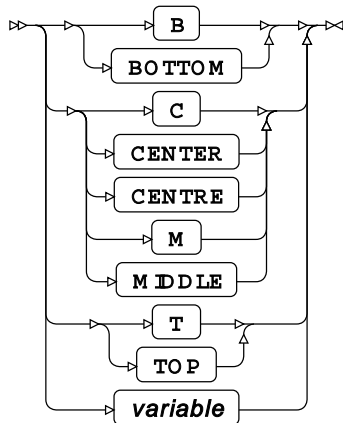
Boolean value



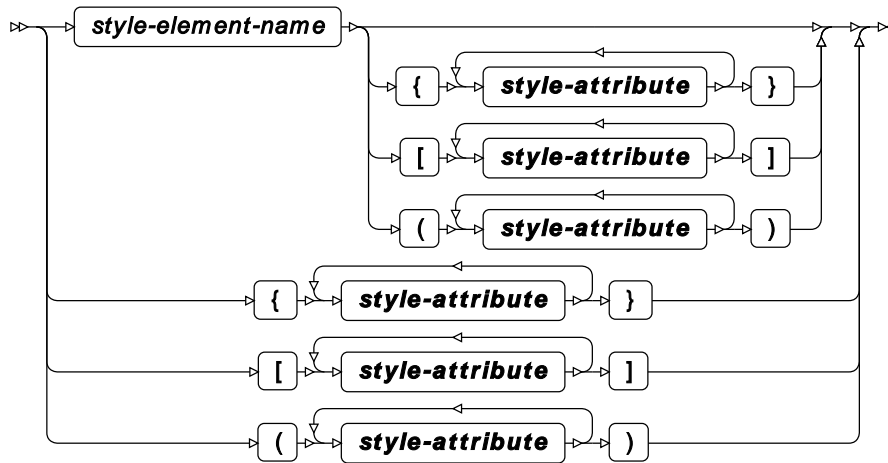
horizontal justification value



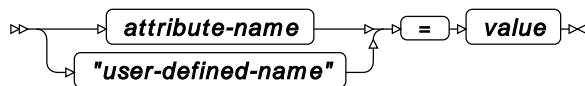
vertical justification value



STYLE



style-attribute



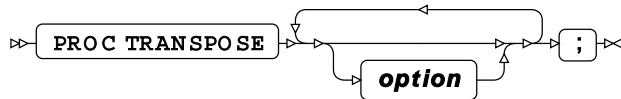
TRANSPOSE procedure

Supported statements

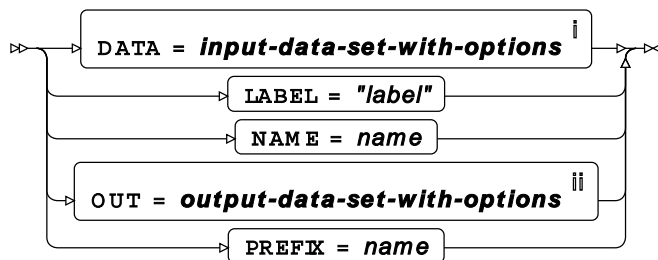
- *PROC TRANSPOSE* [↗](#) (page 2535)
- *BY* [↗](#) (page 2535)
- *COPY* [↗](#) (page 2535)
- *ID* [↗](#) (page 2536)
- *IDLABEL* [↗](#) (page 2536)
- *VAR* [↗](#) (page 2536)
- *WHERE* [↗](#) (page 2536)

PROC TRANSPOSE

Transposes variables (columns) and observations (rows) of a dataset.



option

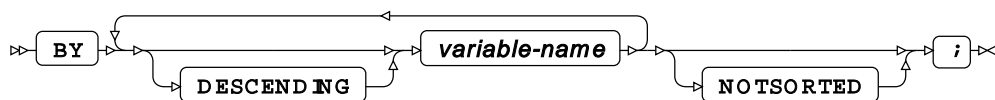


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

BY

Groups the observations in a dataset using one or more specified variables.



COPY

Specifies variables copied untransposed to the output.



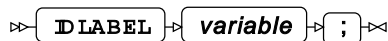
ID

Specifies a variable whose value creates the variable names in the output dataset.



IDLABEL

Specifies a variable whose value creates the variable labels in the output dataset.



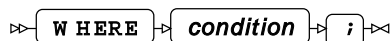
VAR

Specifies the names of variables to be transposed.



WHERE

Restricts the observations to be processed.



TRANTAB procedure

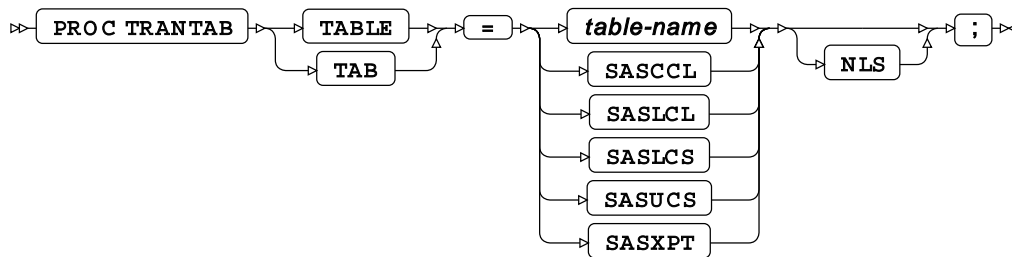
Supported statements

- *PROC TRANTAB* [↗](#) (page 2537)
- *CLEAR* [↗](#) (page 2537)
- *INVERSE* [↗](#) (page 2537)
- *LIST* [↗](#) (page 2538)
- *LOAD* [↗](#) (page 2538)
- *REPLACE* [↗](#) (page 2538)

- [SAVE](#) (page 2538)
- [SWAP](#) (page 2539)

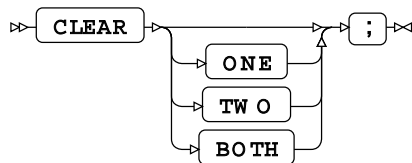
PROC TRANTAB

Edits translation tables that control the conversion between specific single-byte character sets.



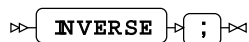
CLEAR

Clears one or both of the currently loaded translation tables.



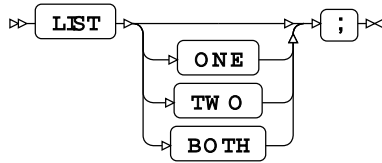
INVERSE

Populates table two with the inverse of the translation in table one.



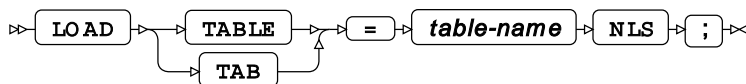
LIST

Prints out table one or both of the currently loaded translation tables.



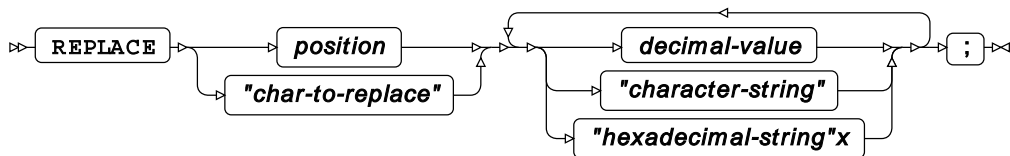
LOAD

Loads translation tables from a catalog.



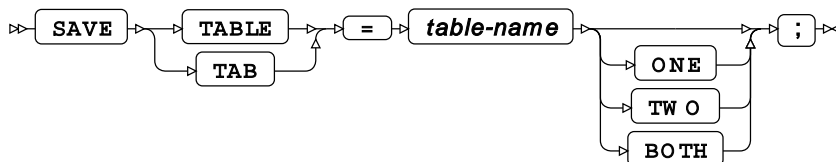
REPLACE

Replaces an entry in a translation table.



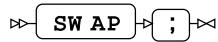
SAVE

Saves one or both translation tables to the original location, or a new location.



SWAP

Swaps definitions of table one and table two.



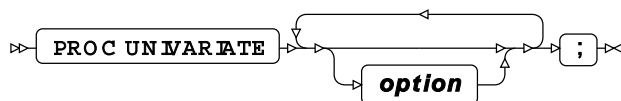
UNIVARIATE procedure

Supported statements

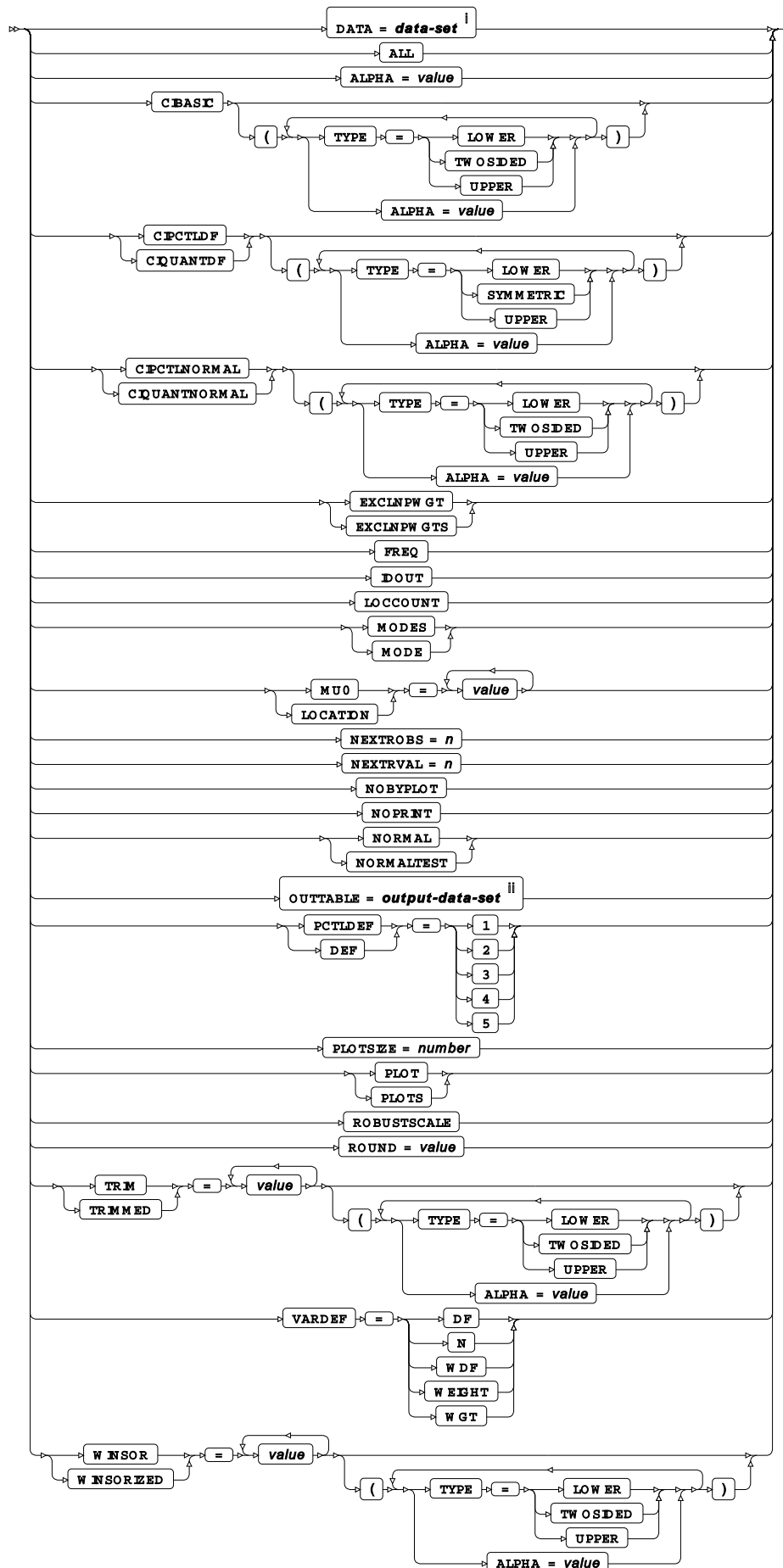
- *PROC UNIVARIATE* [↗](#) (page 2539)
- BY [↗](#) (page 2541)
- CDFPLOT [↗](#) (page 2541)
- CLASS [↗](#) (page 2547)
- FREQ [↗](#) (page 2547)
- HISTOGRAM [↗](#) (page 2548)
- ID [↗](#) (page 2558)
- OUTPUT [↗](#) (page 2558)
- PPLOT [↗](#) (page 2561)
- PROBLOT [↗](#) (page 2567)
- QQPLOT [↗](#) (page 2573)
- VAR [↗](#) (page 2579)
- WEIGHT [↗](#) (page 2579)
- WHERE [↗](#) (page 2579)

PROC UNIVARIATE

Generates univariate statistics and plots for variables in a dataset.



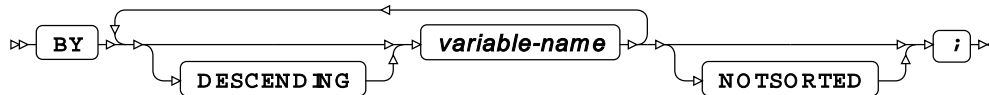
option



- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).

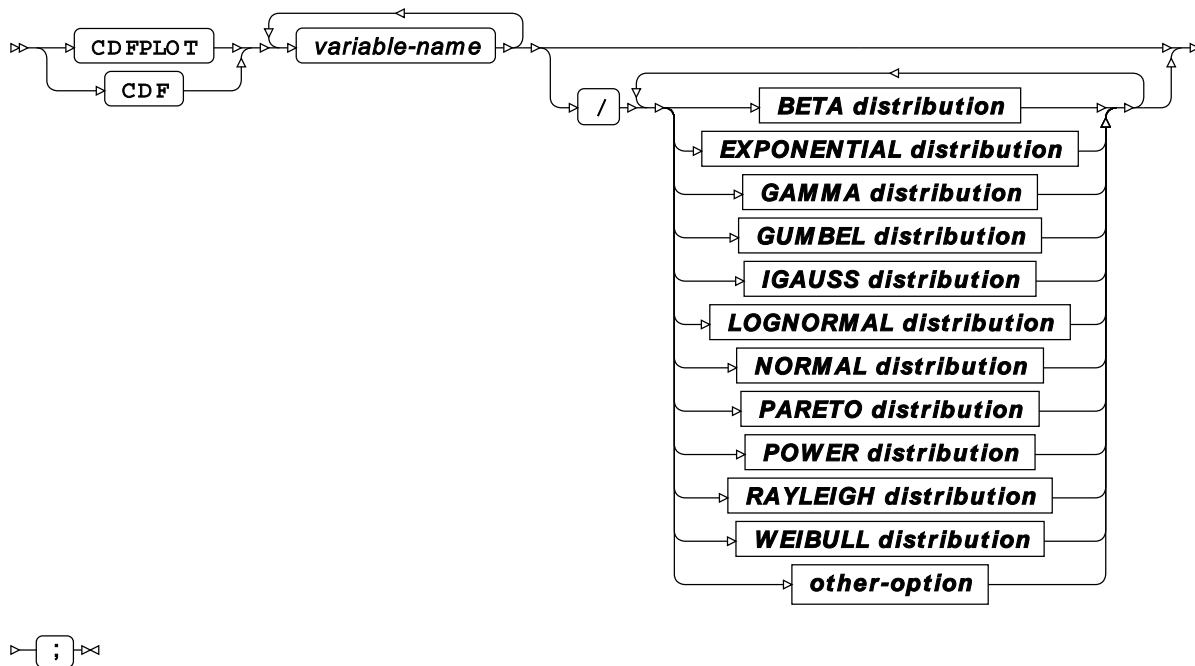
BY

Groups the observations in a dataset using one or more specified variables.

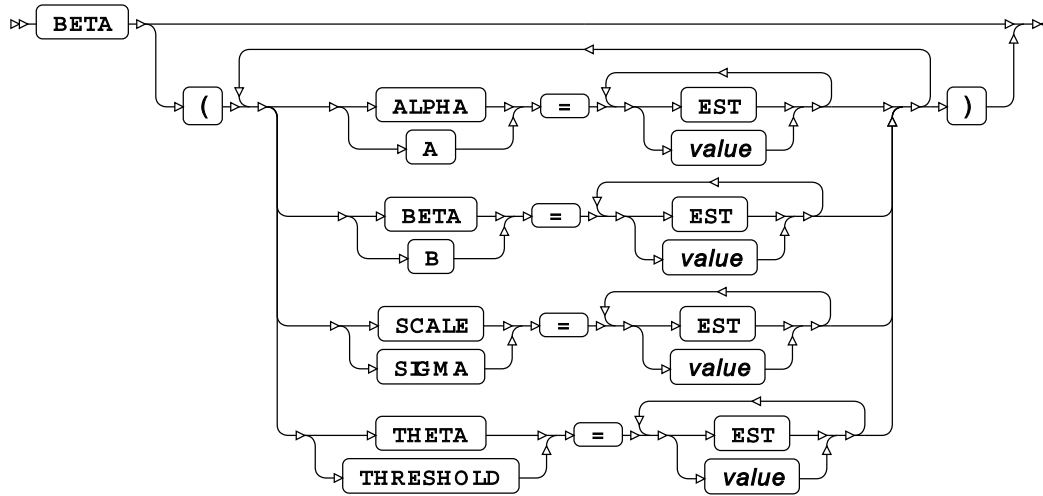


CDFPLOT

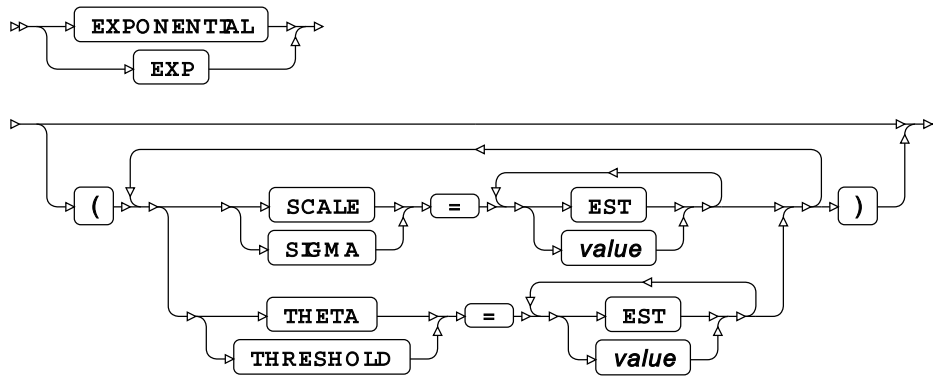
Plots the cumulative distribution function with options.



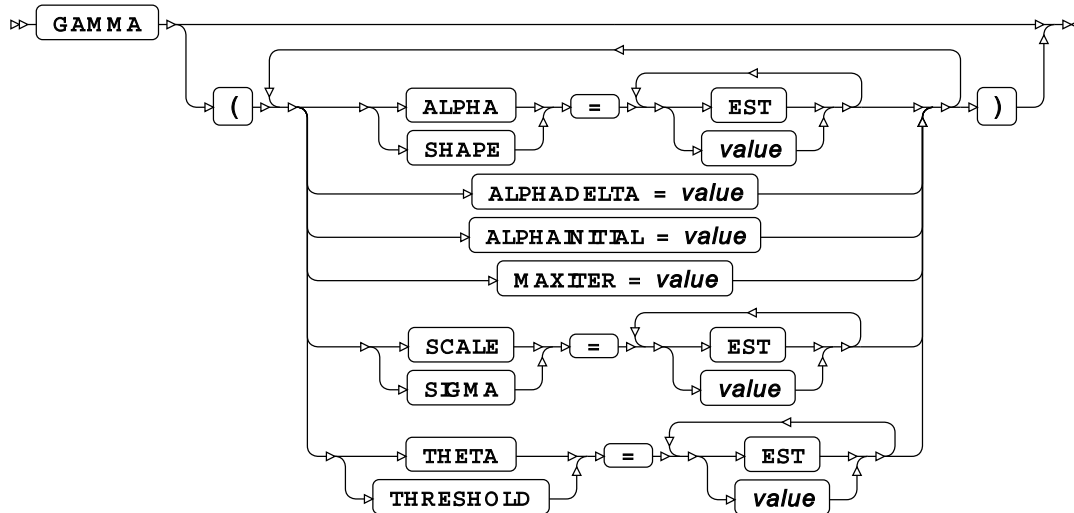
BETA distribution



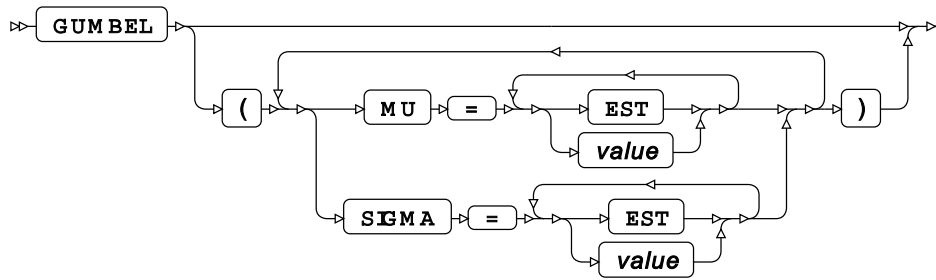
EXPONENTIAL distribution



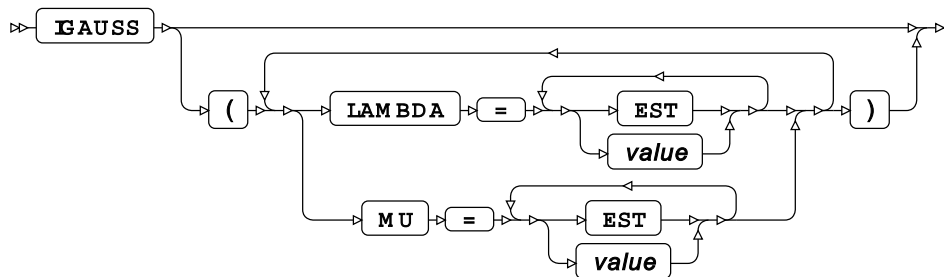
GAMMA distribution



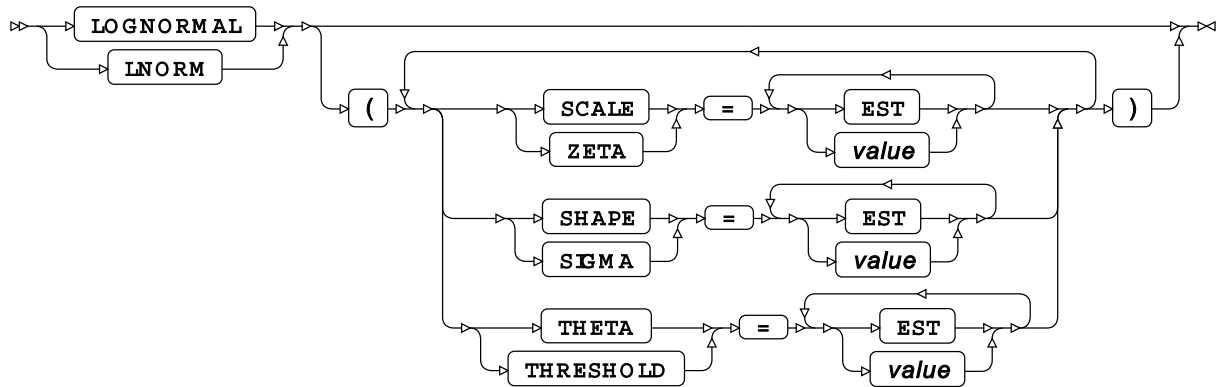
GUMBEL distribution



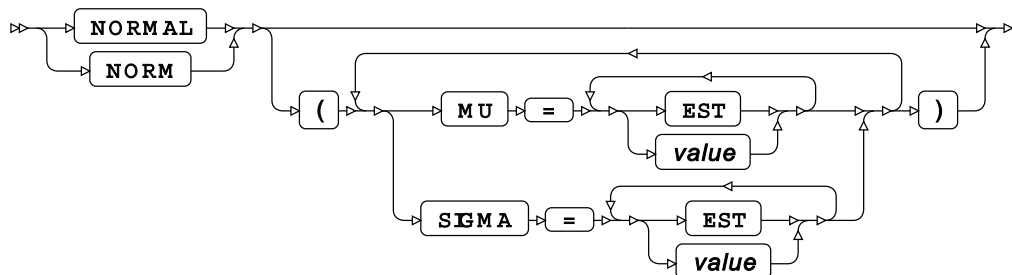
IGAUSS distribution



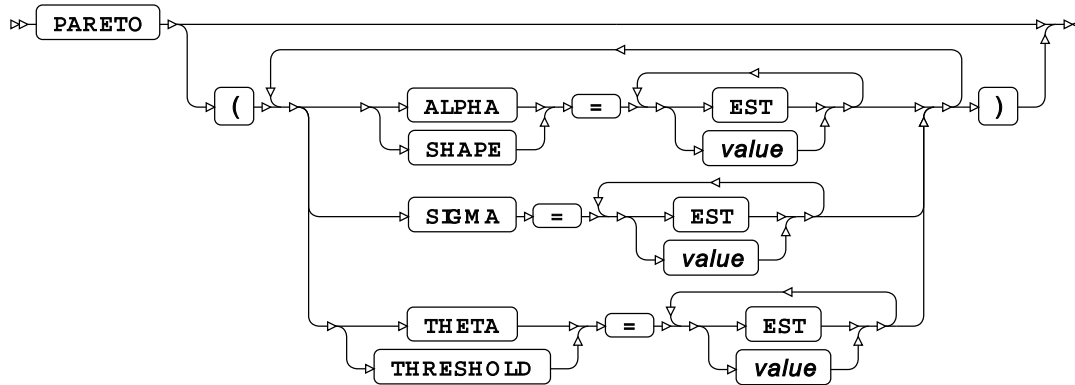
LOGNORMAL distribution



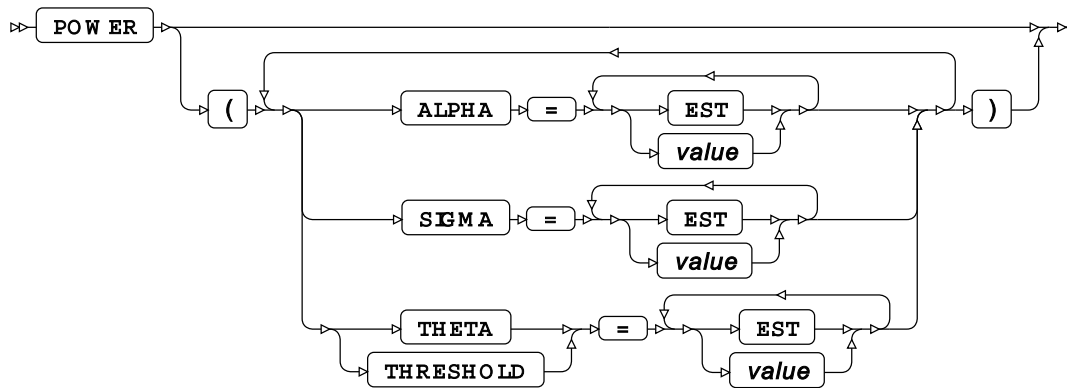
NORMAL distribution



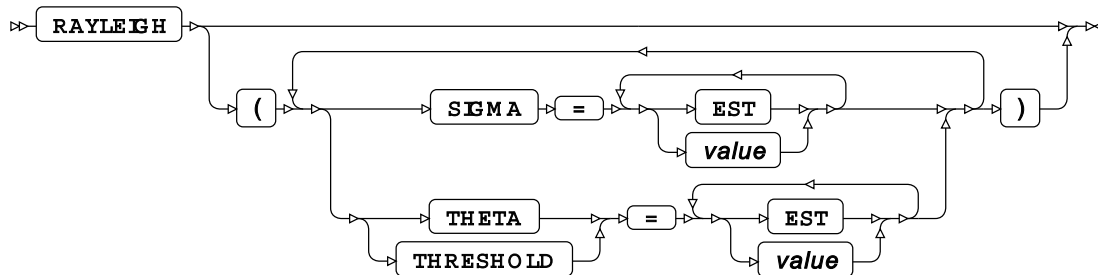
PARETO distribution



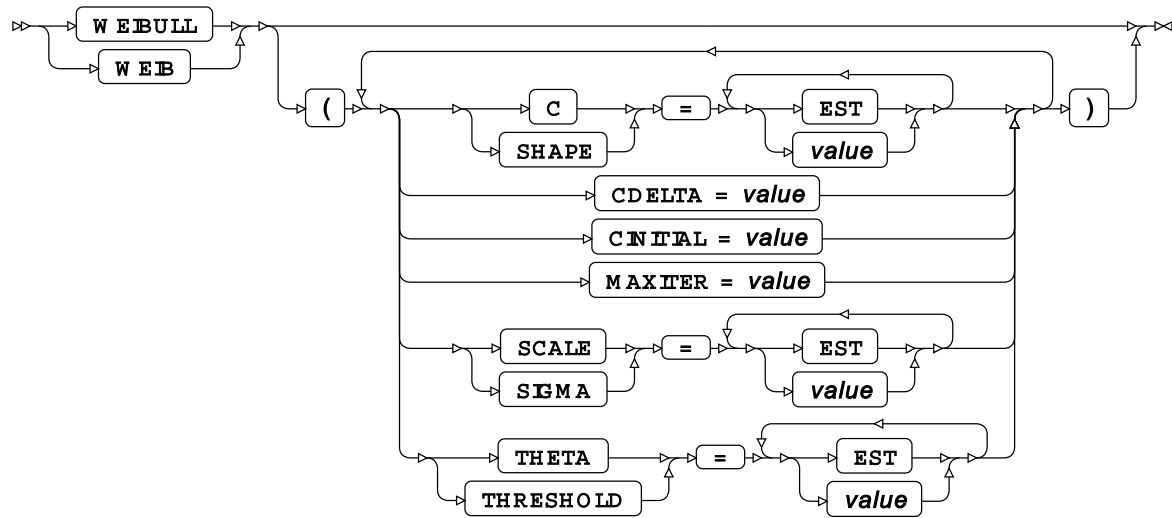
POWER distribution



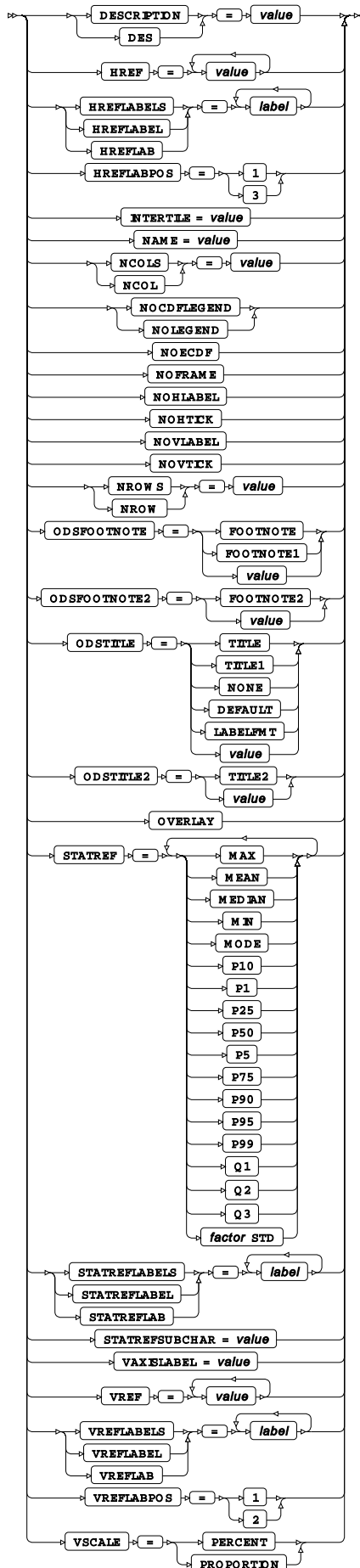
RAYLEIGH distribution



WEIBULL distribution

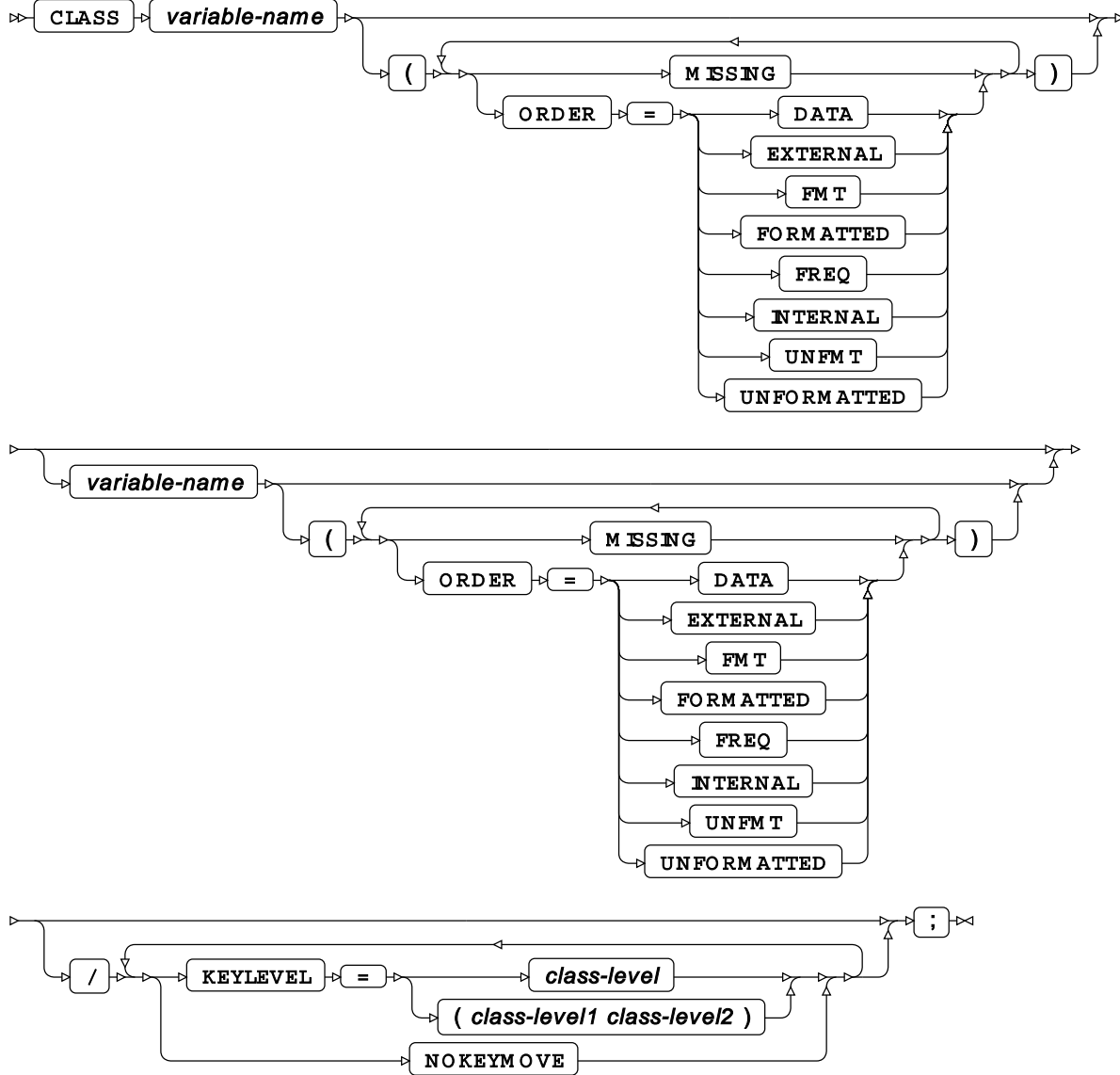


other-option



CLASS

Specifies variables used to group the data in plots.



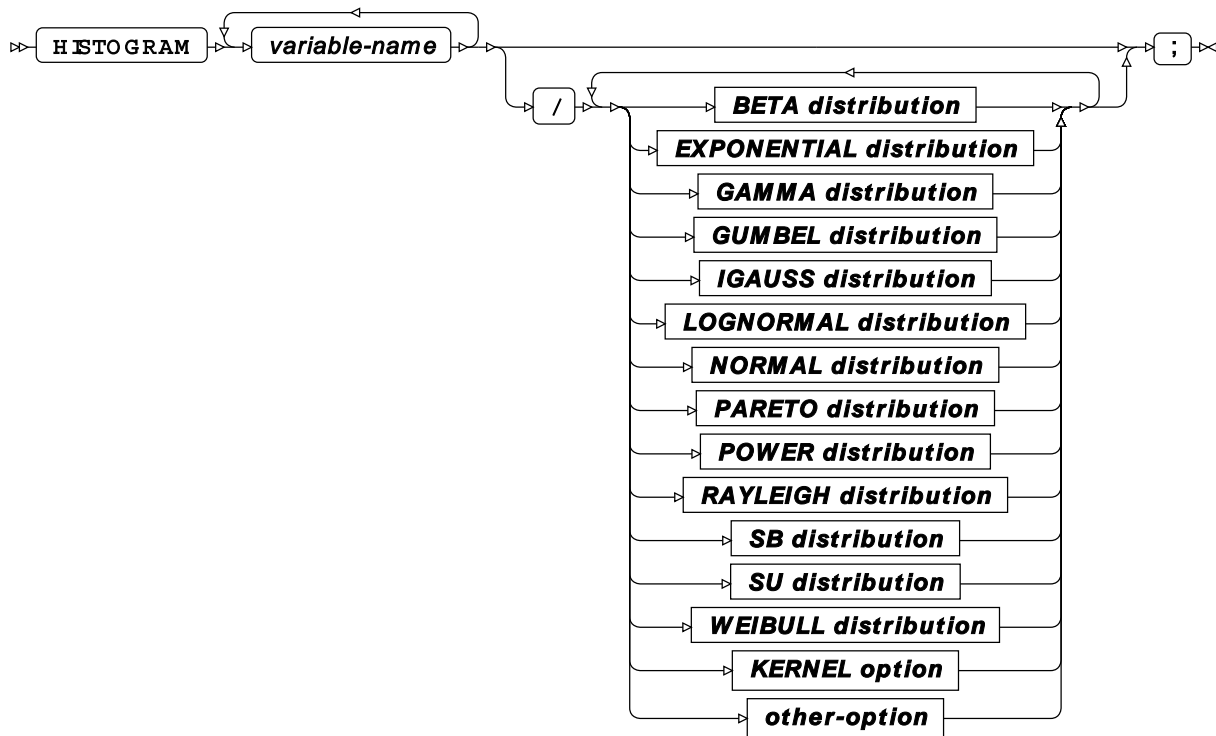
FREQ

Specifies a variable in which the frequency to be associated with each observation is provided.

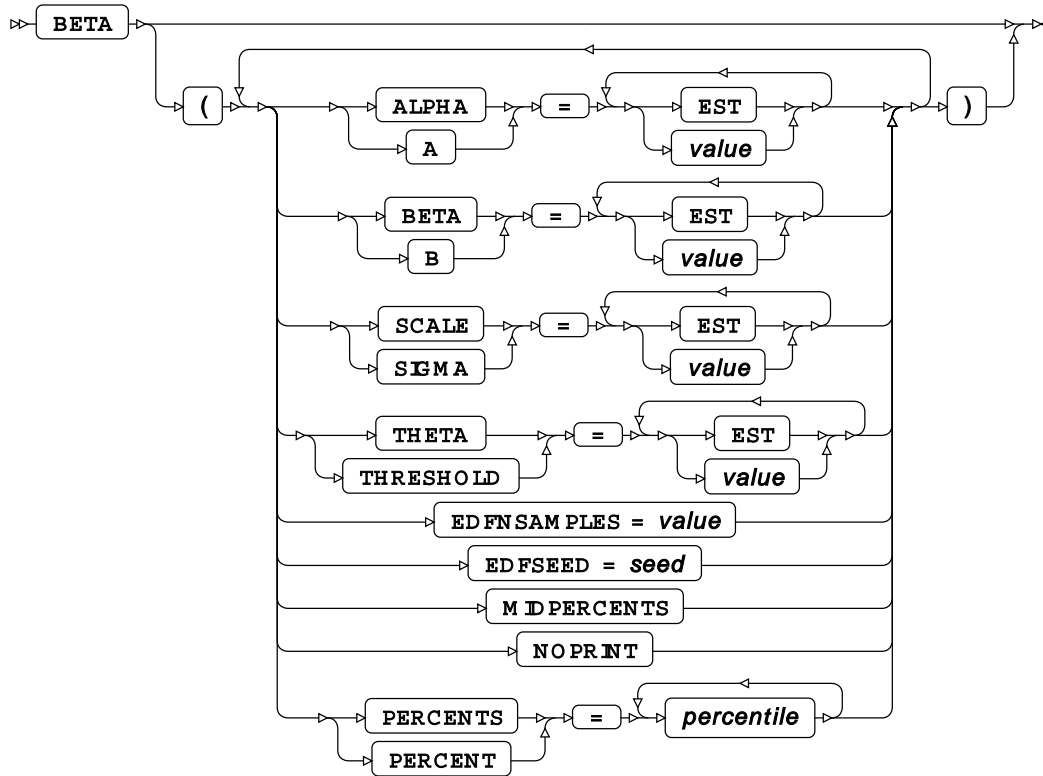


HISTOGRAM

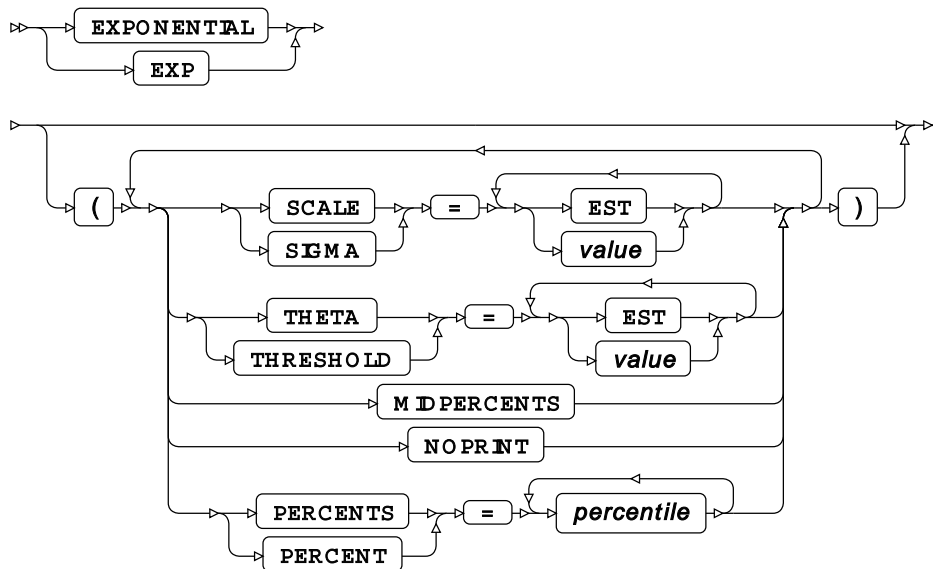
Creates a histogram.



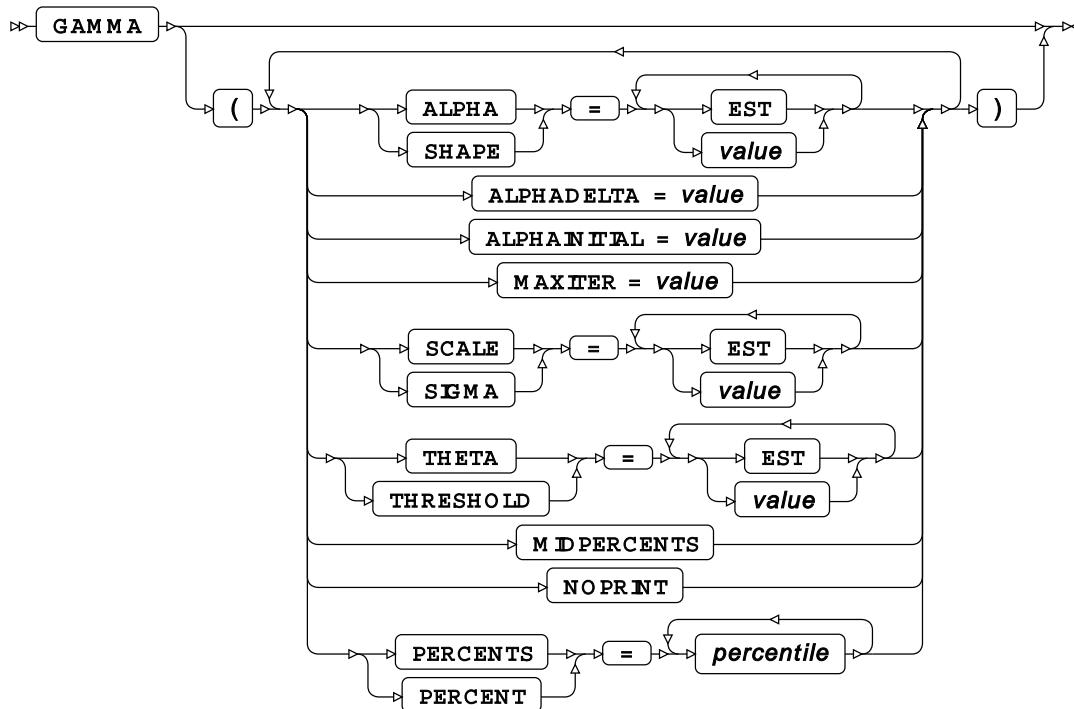
BETA distribution



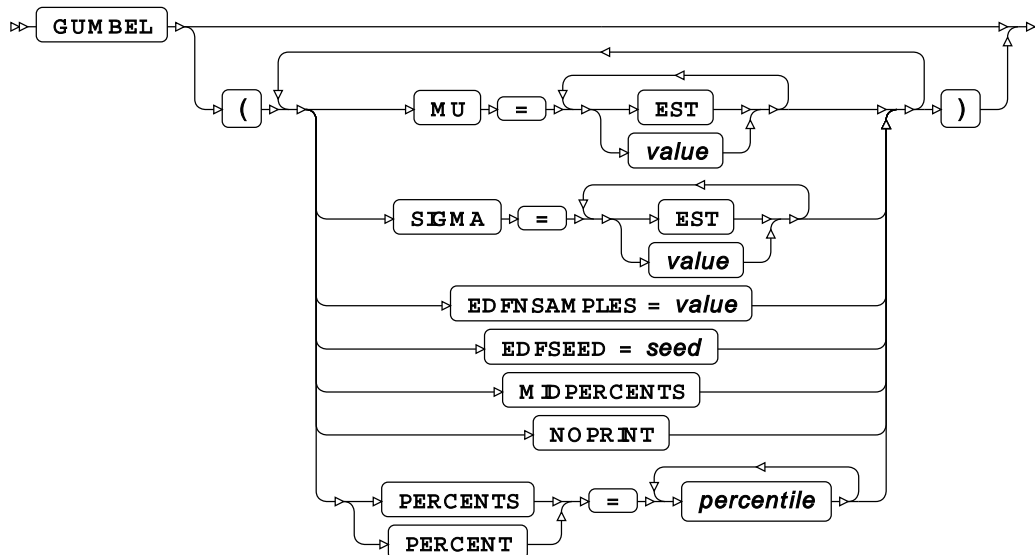
EXPONENTIAL distribution



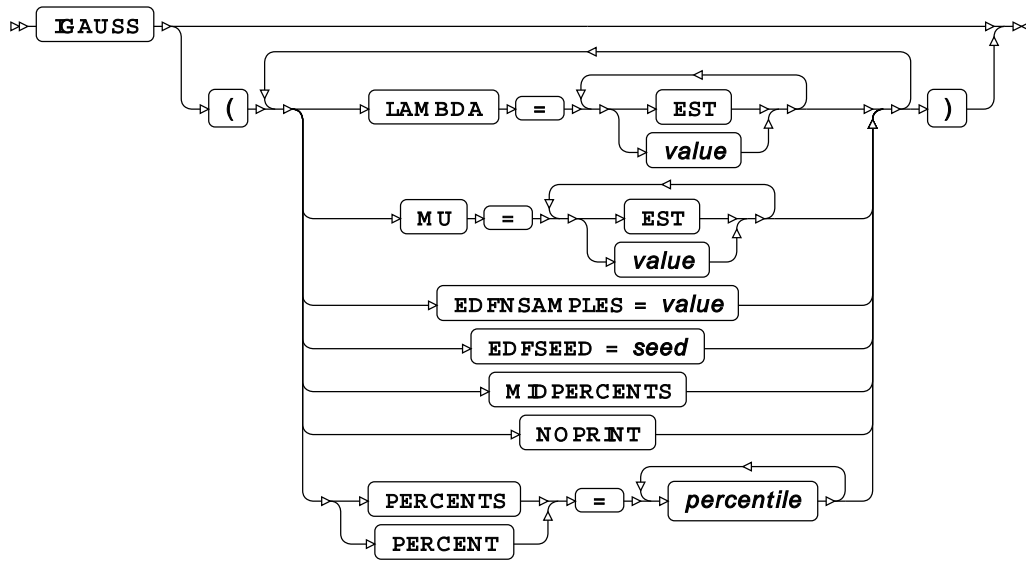
GAMMA distribution



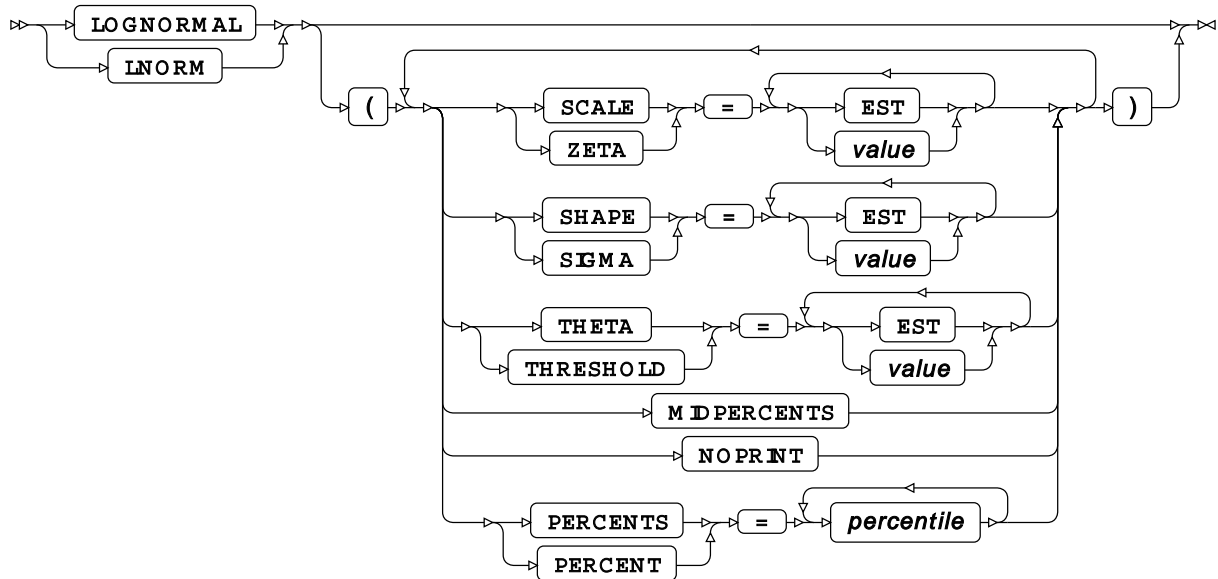
GUMBEL distribution



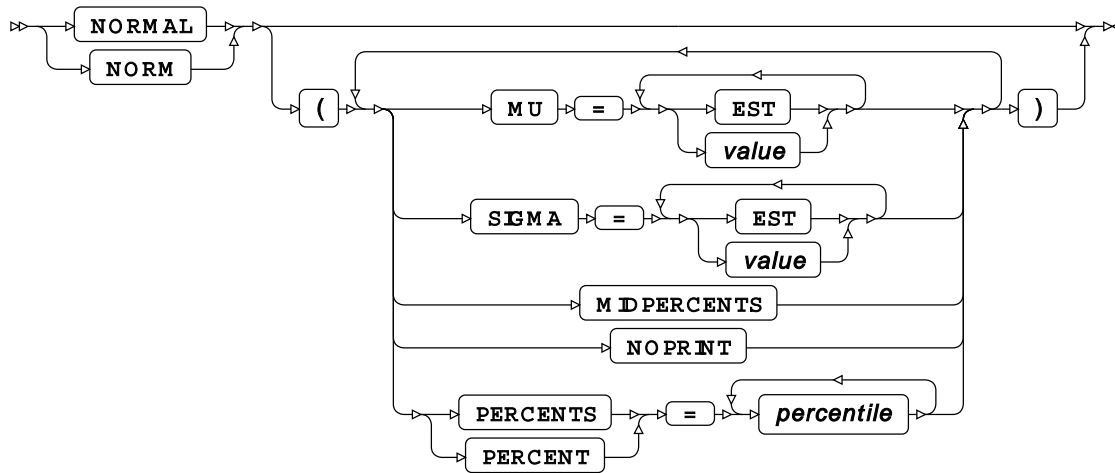
IGAUSS distribution



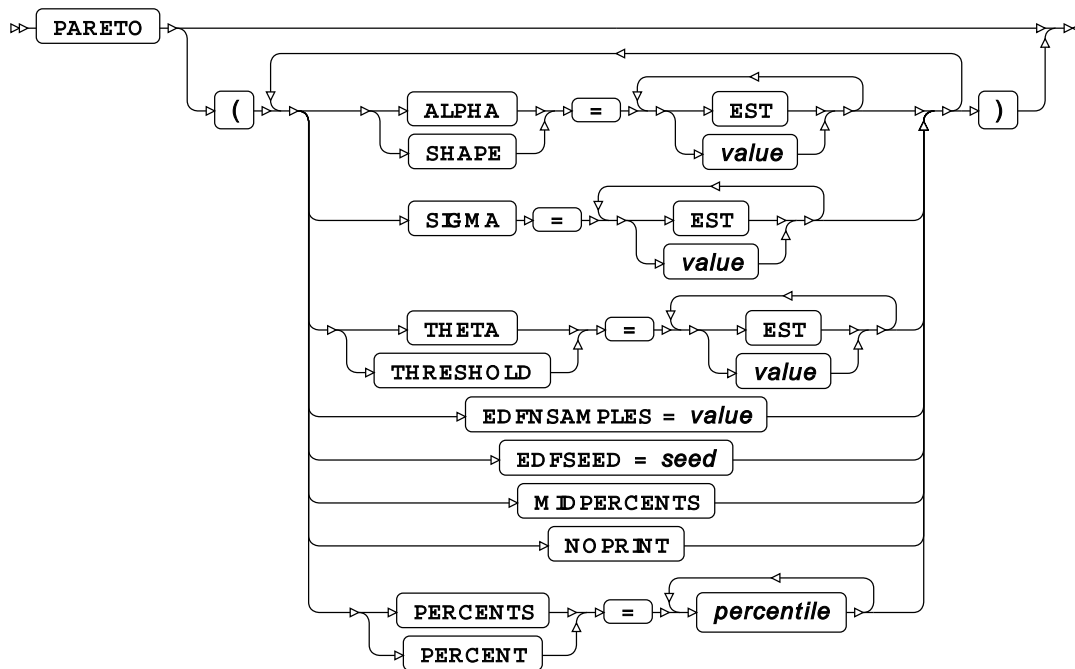
LOGNORMAL distribution



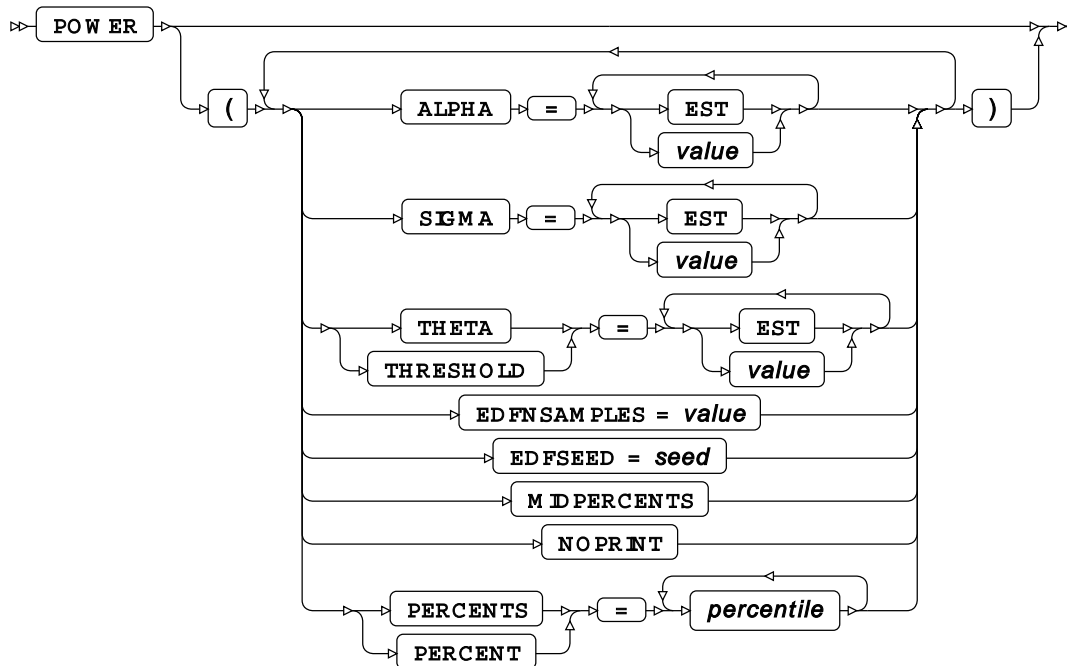
NORMAL distribution



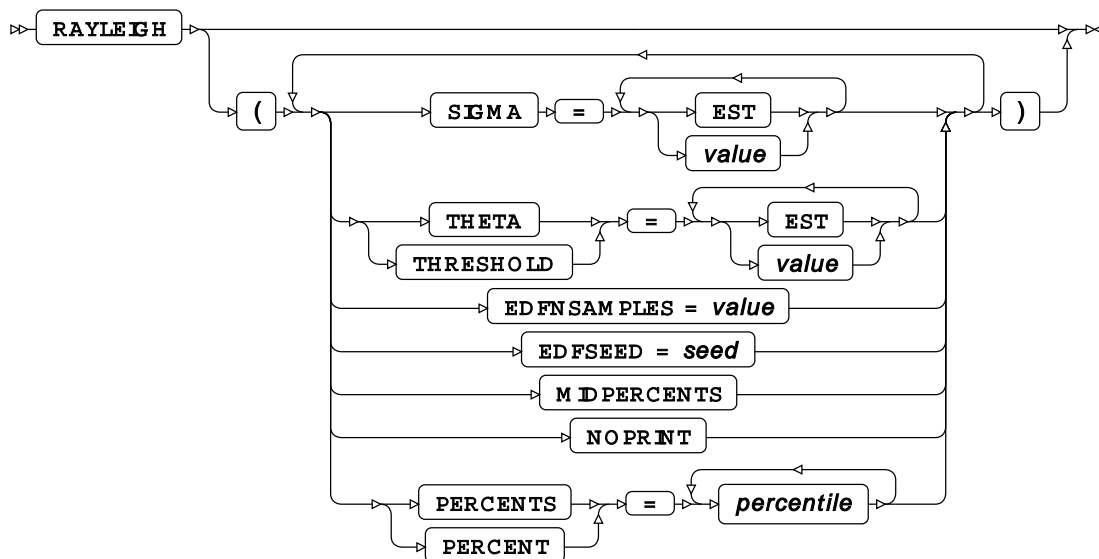
PARETO distribution



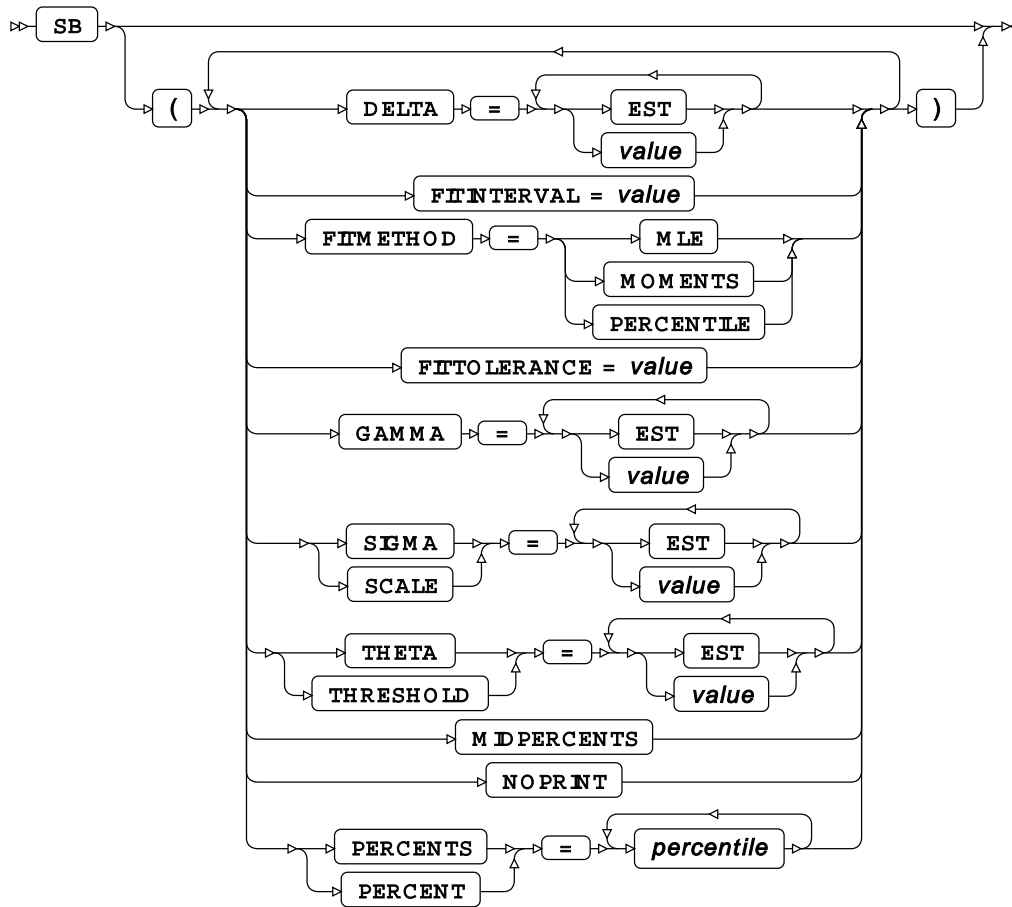
POWER distribution



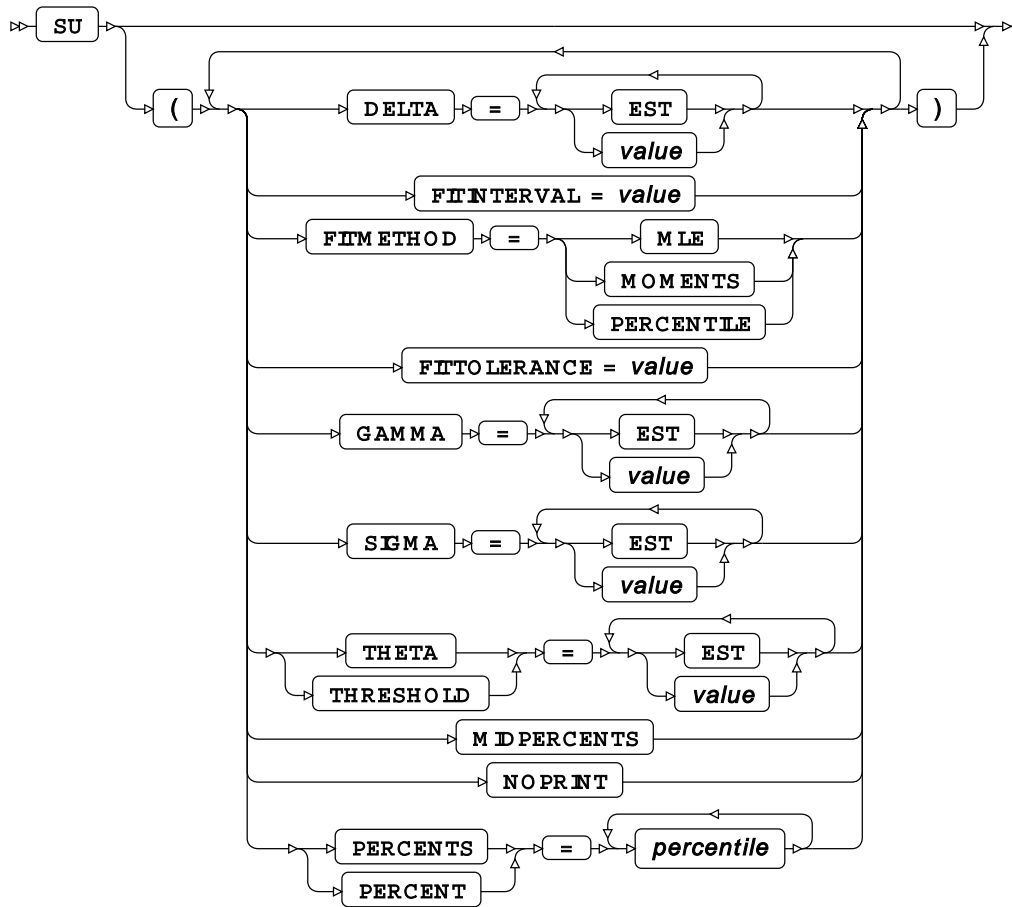
RAYLEIGH distribution



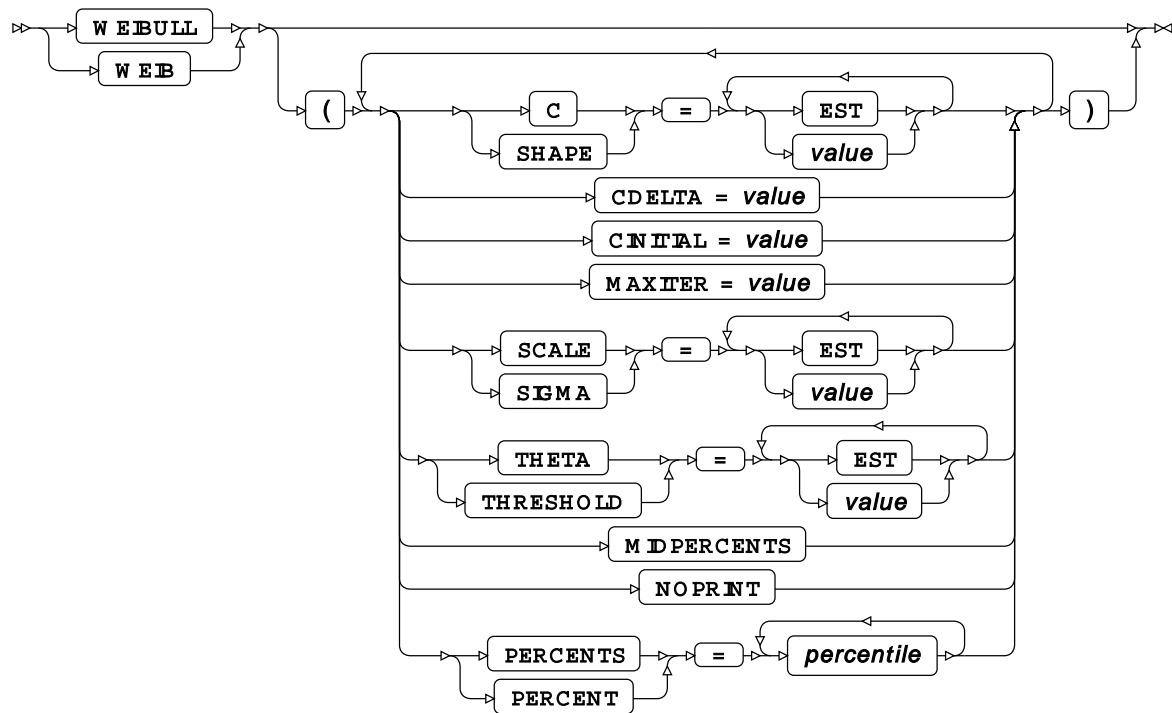
SB distribution



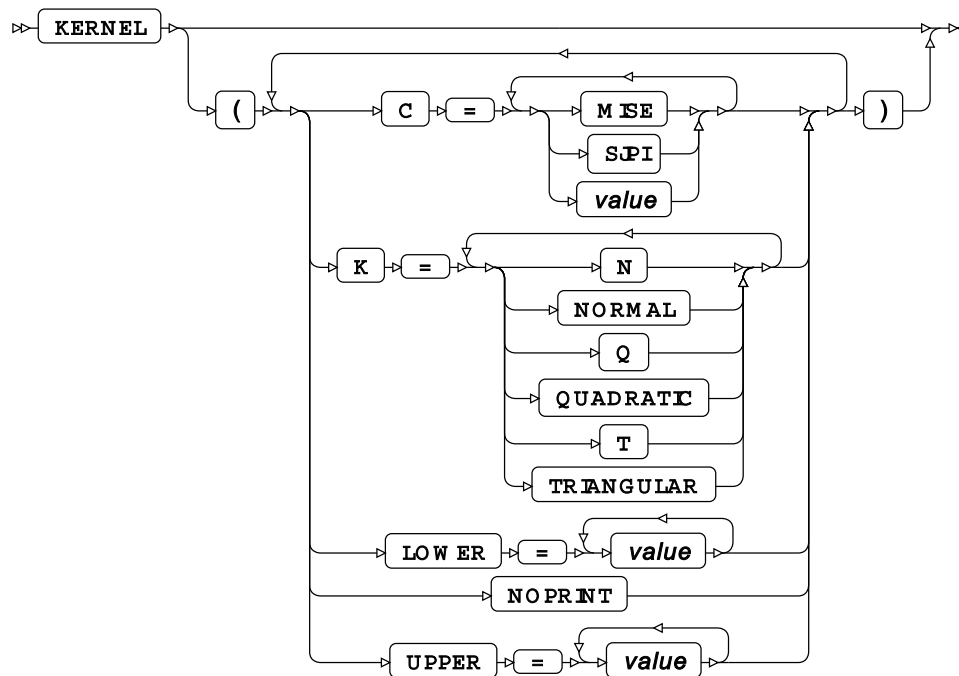
SU distribution



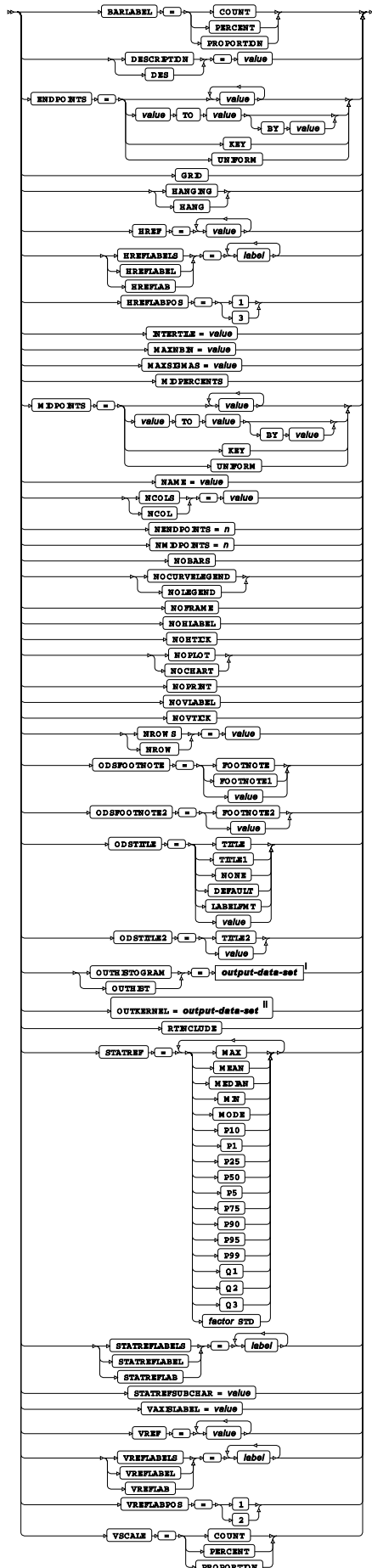
WEIBULL distribution



KERNEL option



other-option

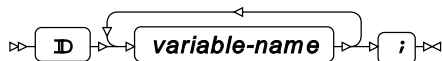


ⁱ See *Output dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

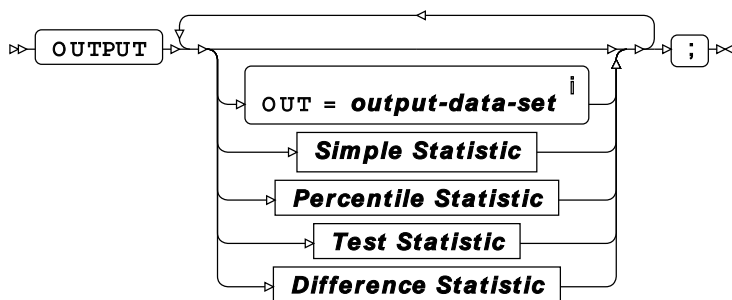
ID

Identifies the relevant observations in the output by using one or more specified variable names.



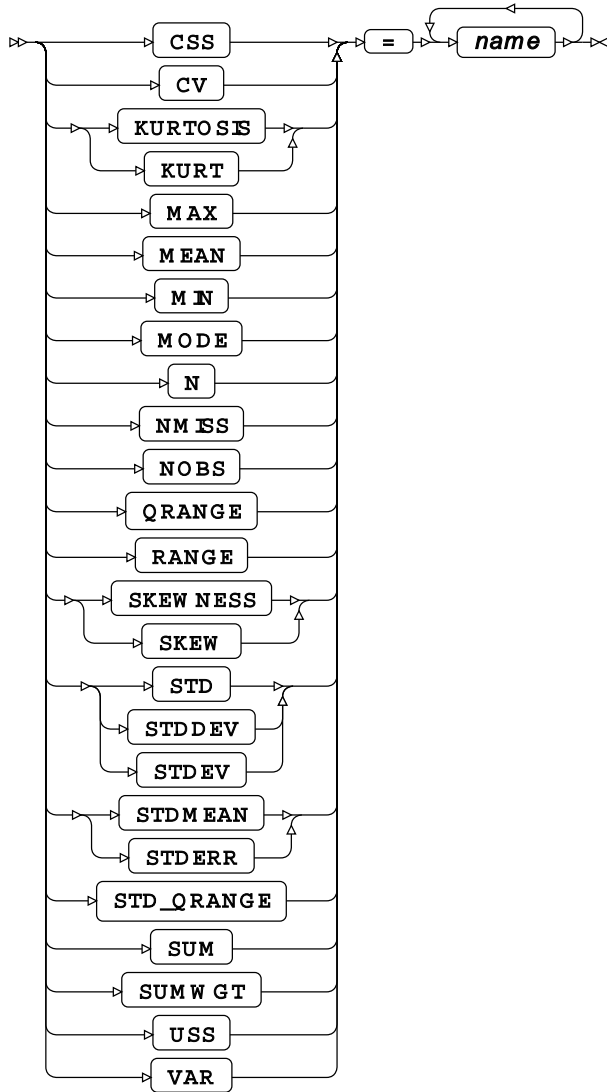
OUTPUT

Saves calculated statistics in an output dataset.

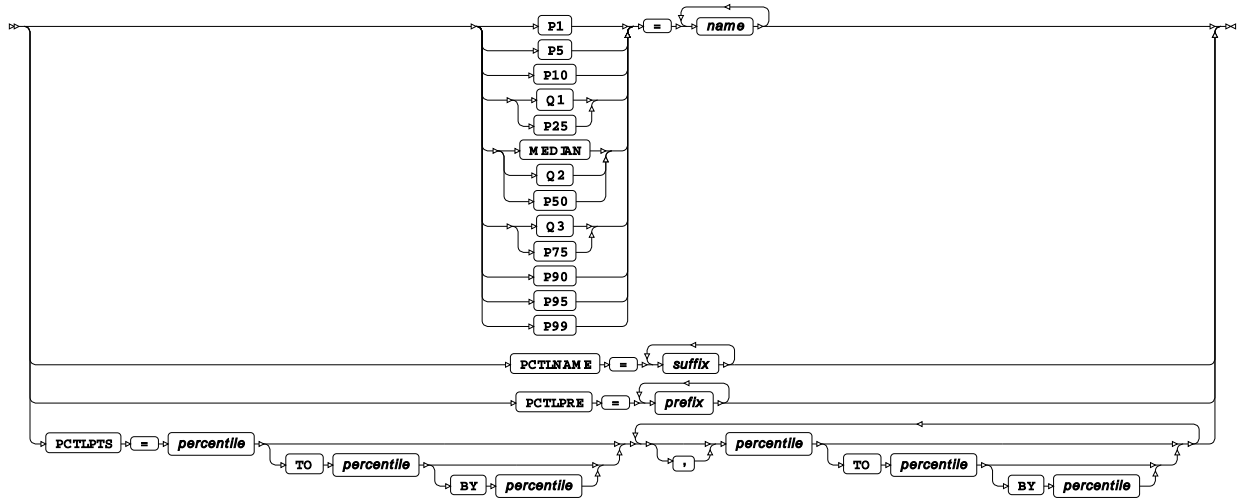


ⁱ See *Output dataset* [↗](#) (page 16).

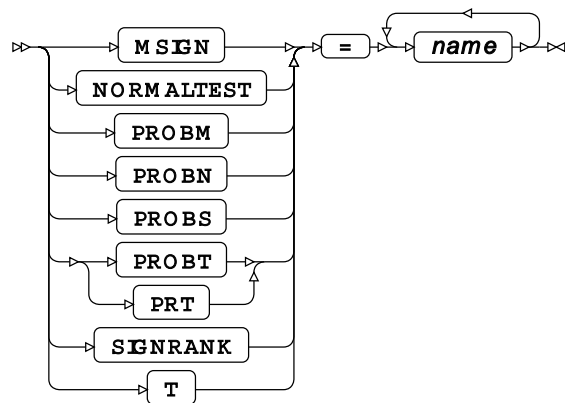
Simple Statistic



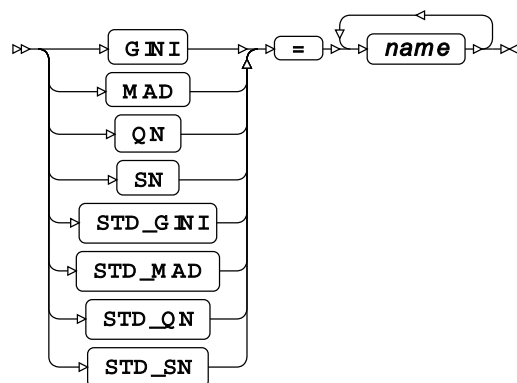
Percentile Statistic



Test Statistic

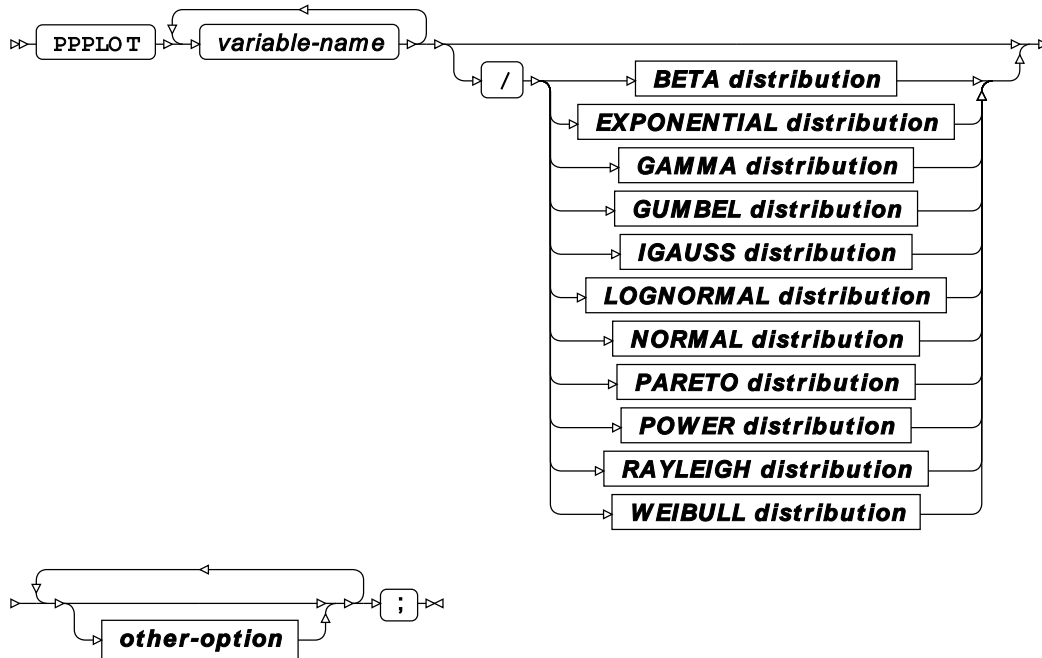


Difference Statistic

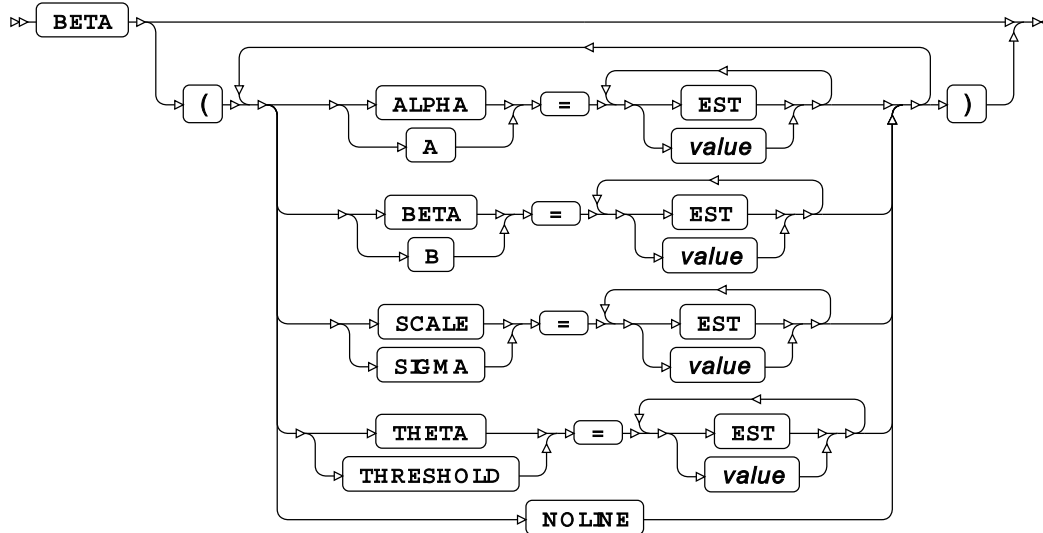


PPPLOT

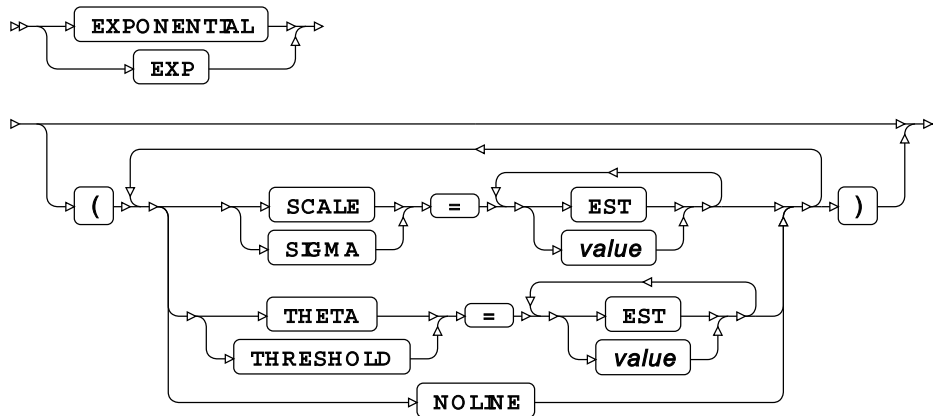
Creates probability percentage plots of datasets using one or more specified variables.



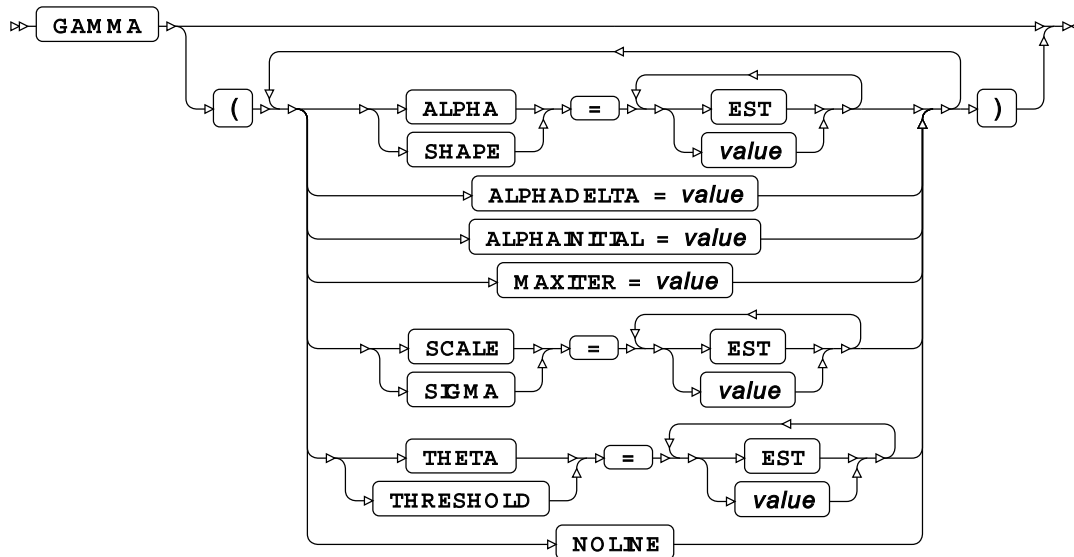
BETA distribution



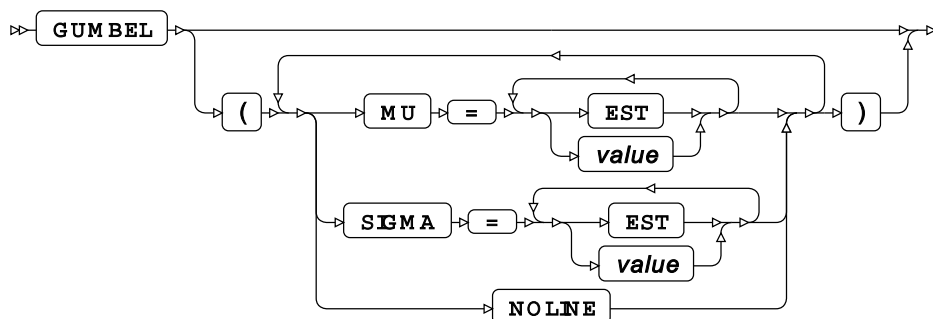
EXPONENTIAL distribution



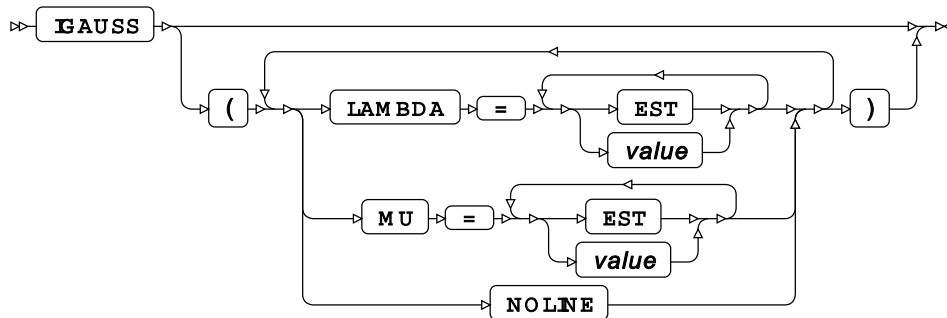
GAMMA distribution



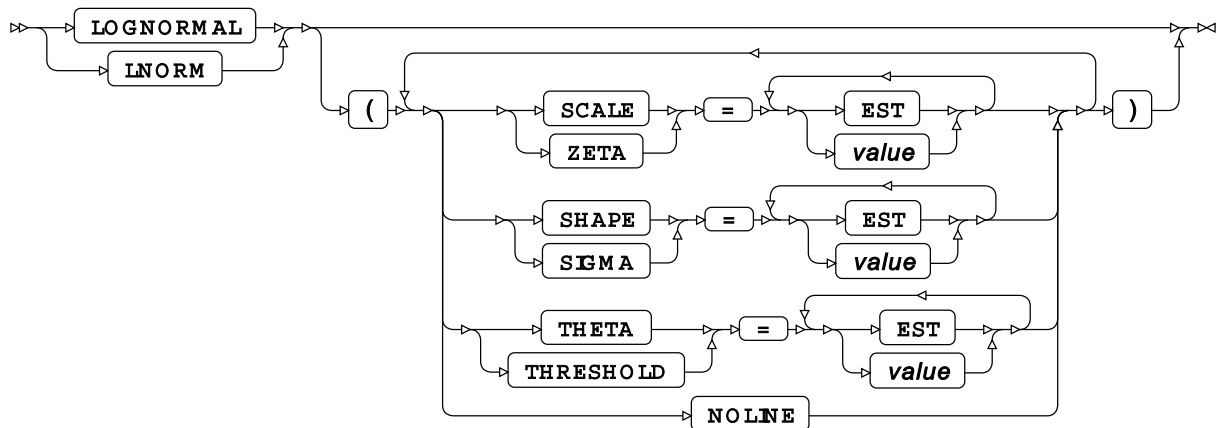
GUMBEL distribution



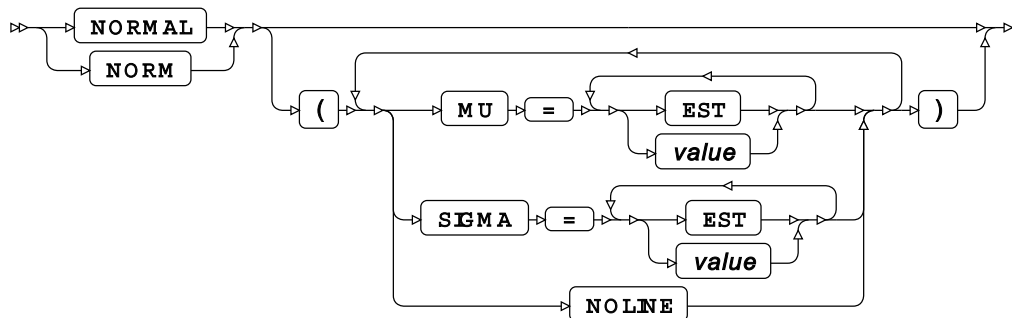
IGAUSS distribution



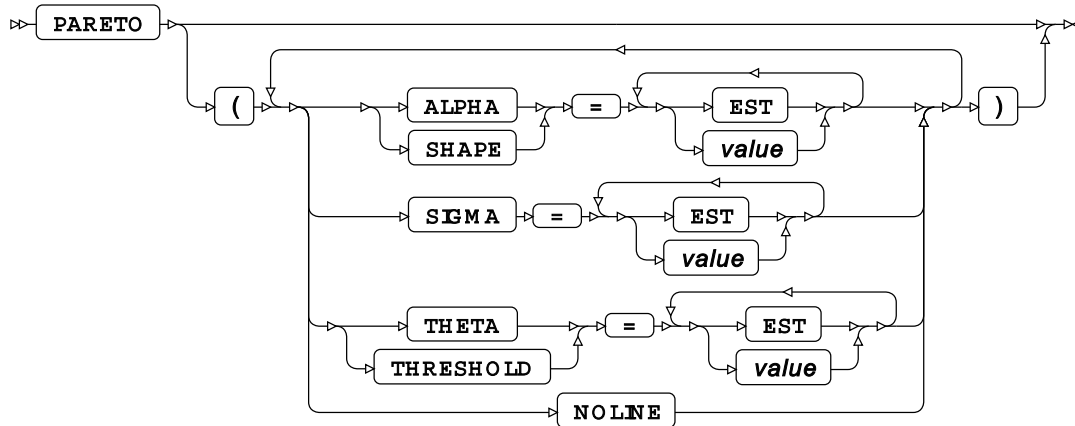
LOGNORMAL distribution



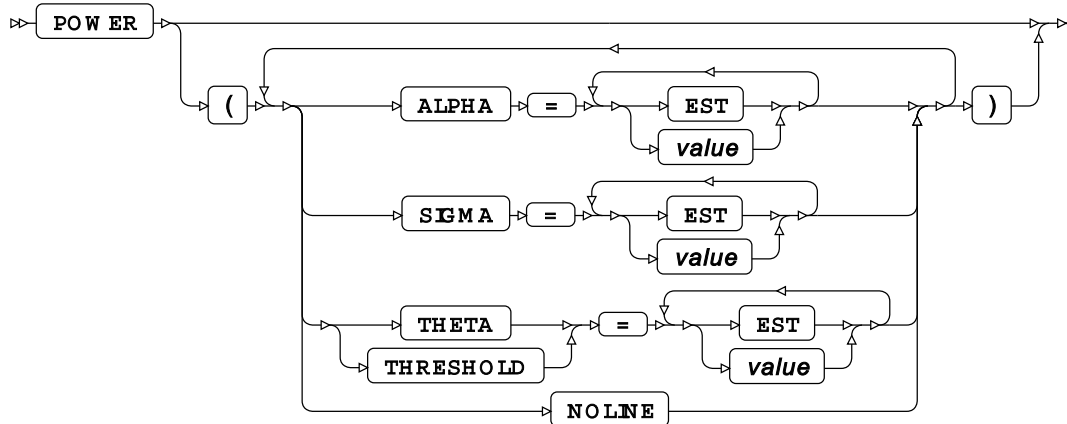
NORMAL distribution



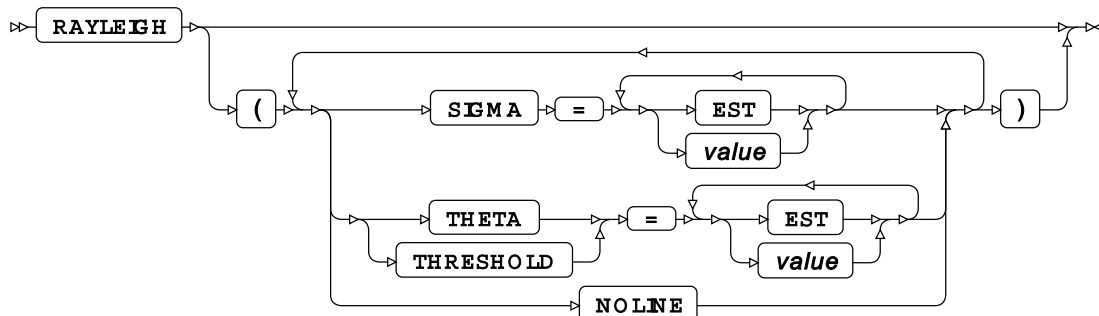
PARETO distribution



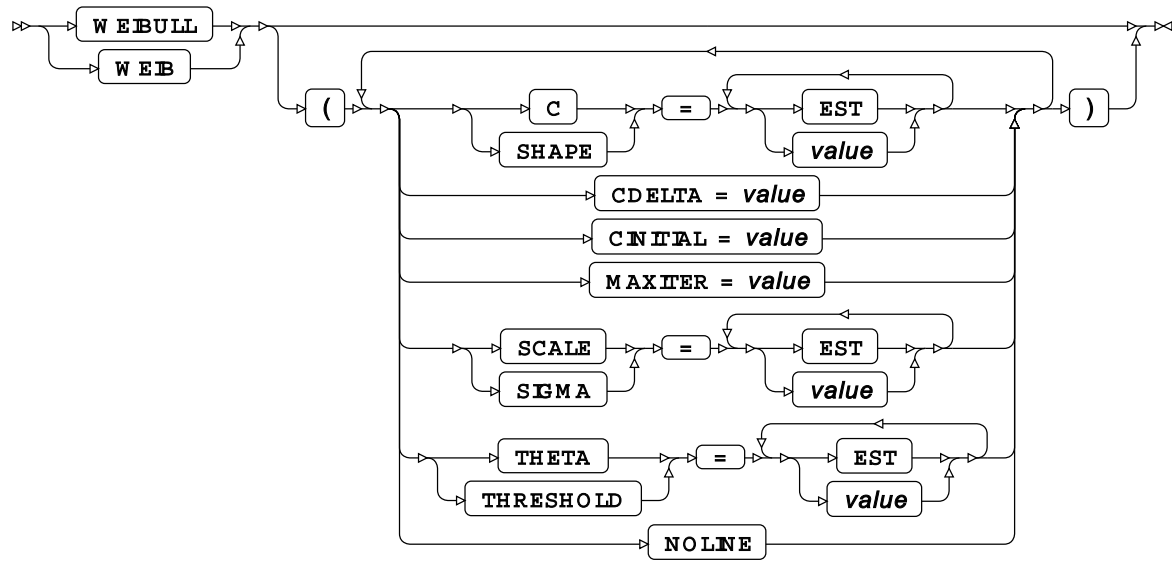
POWER distribution



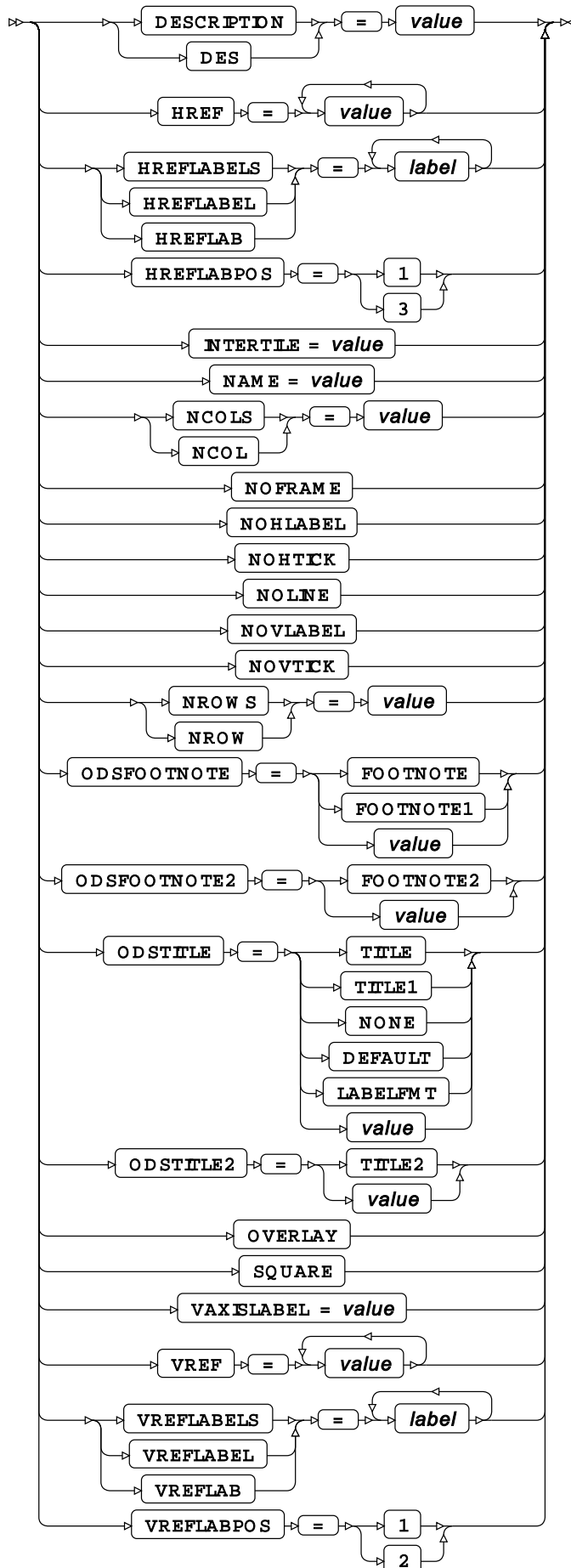
RAYLEIGH distribution



WEIBULL distribution

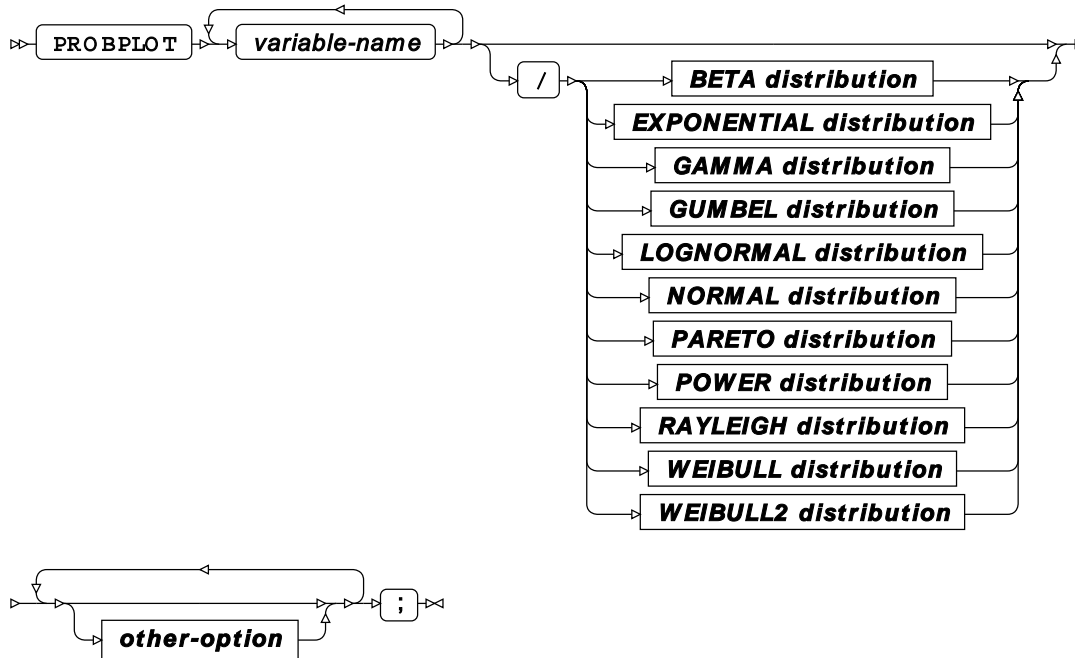


other-option

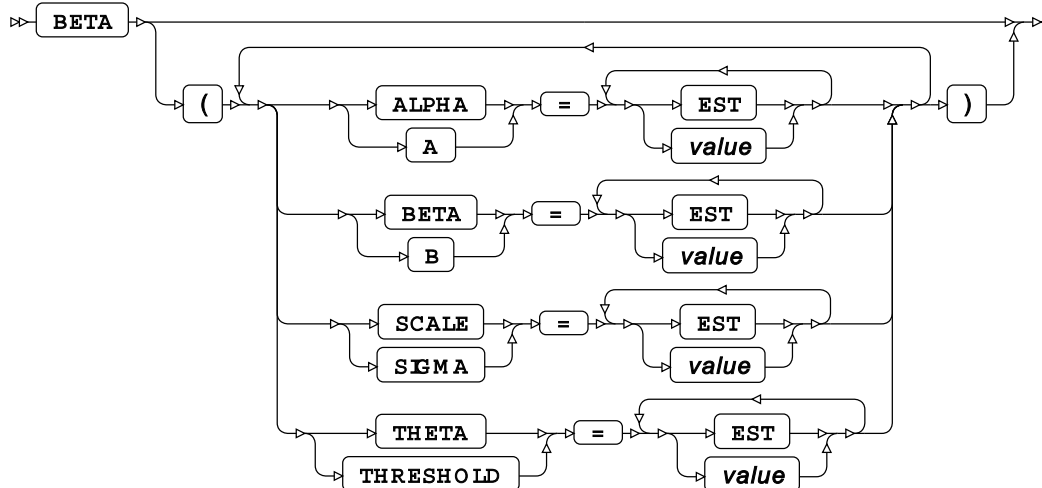


PROBPLOT

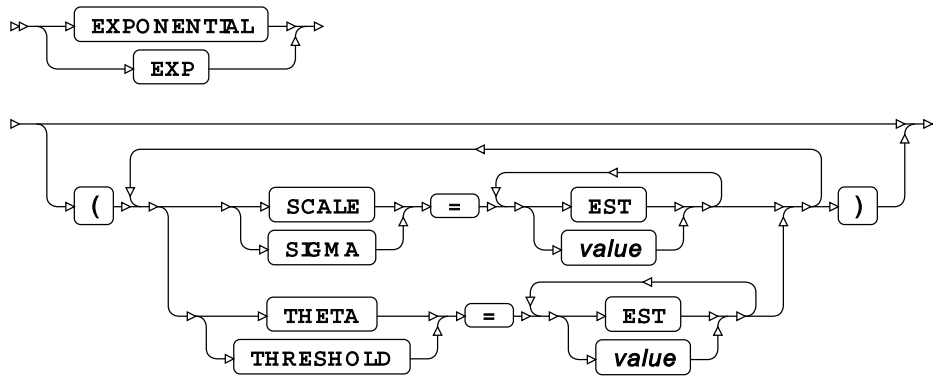
Compares a dataset against a normal distribution using one or more specified variables.



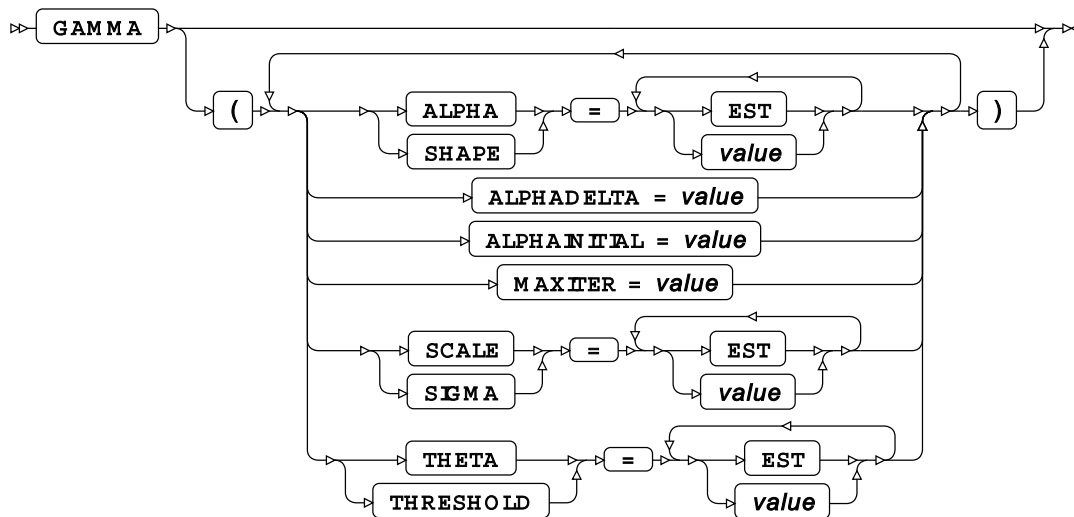
BETA distribution



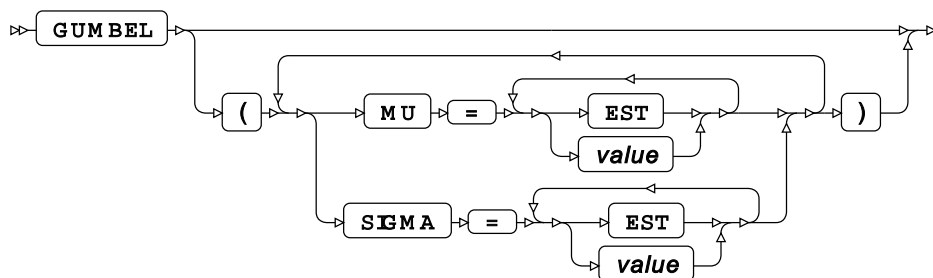
EXPONENTIAL distribution



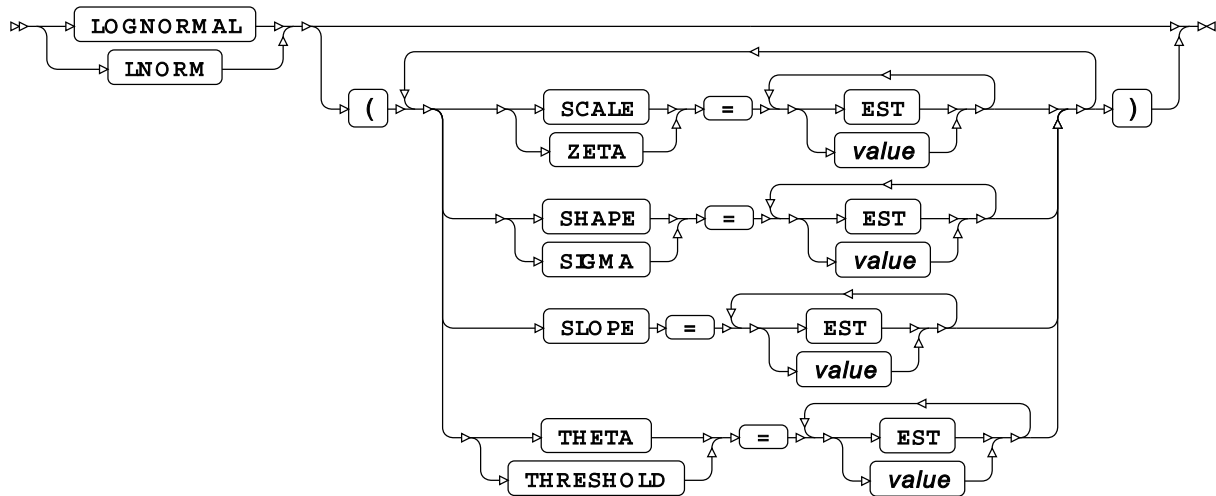
GAMMA distribution



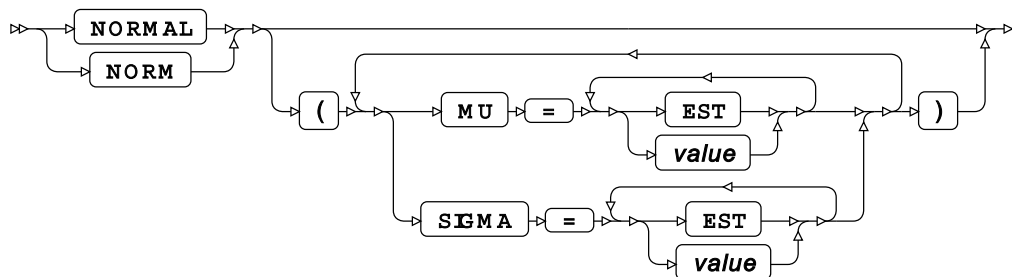
GUMBEL distribution



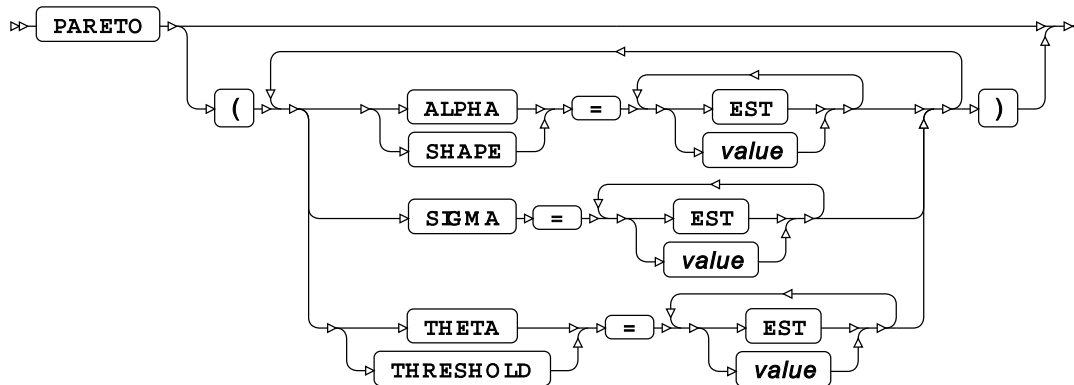
LOGNORMAL distribution



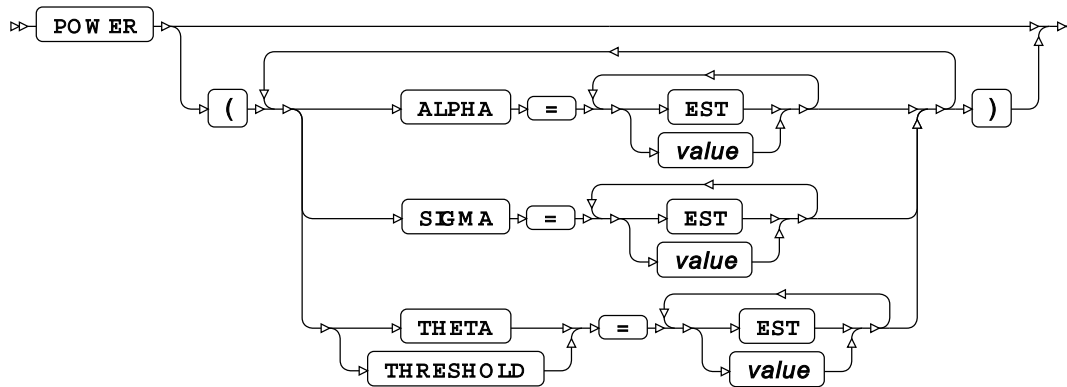
NORMAL distribution



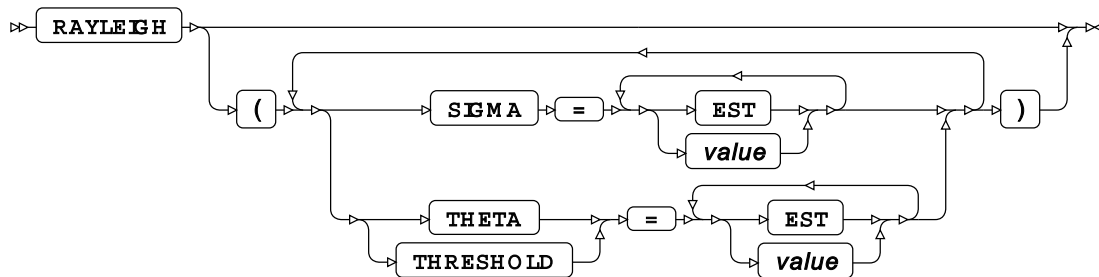
PARETO distribution



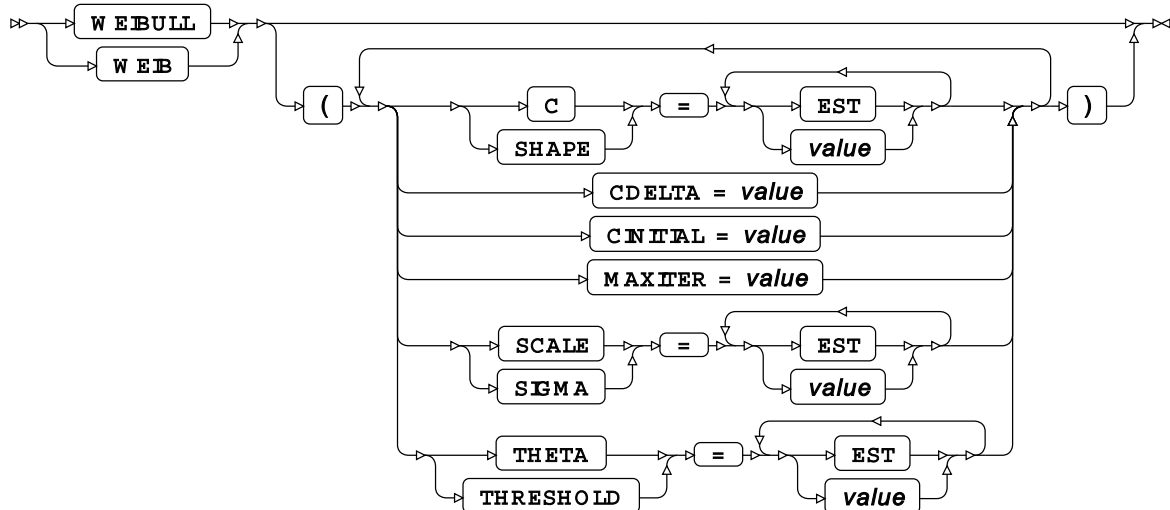
POWER distribution



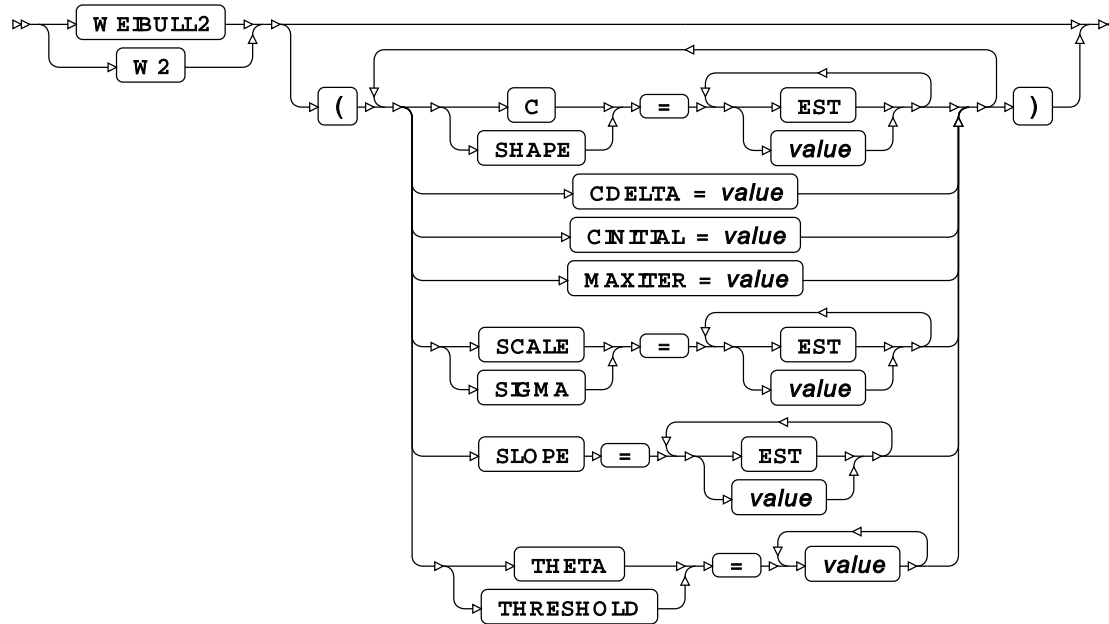
RAYLEIGH distribution



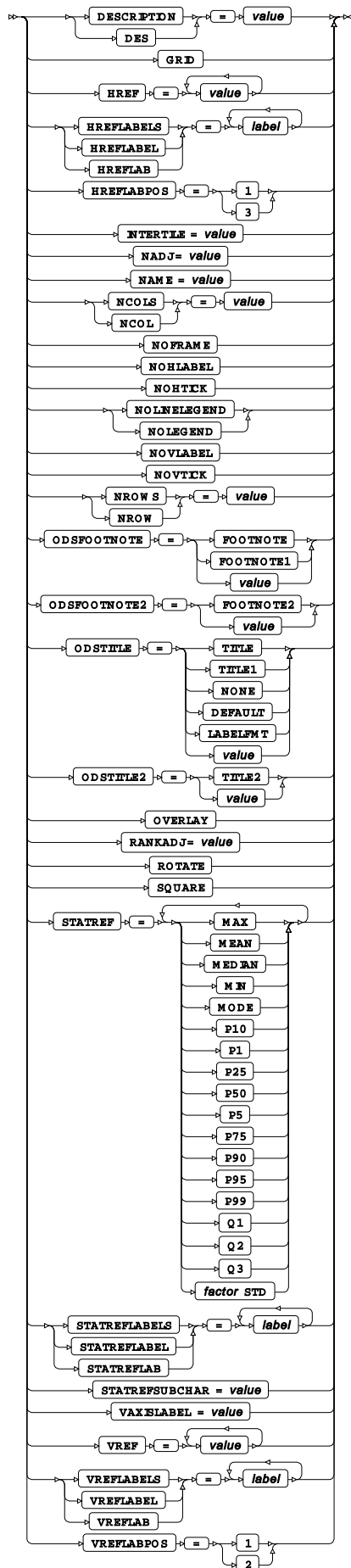
WEIBULL distribution



WEIBULL2 distribution

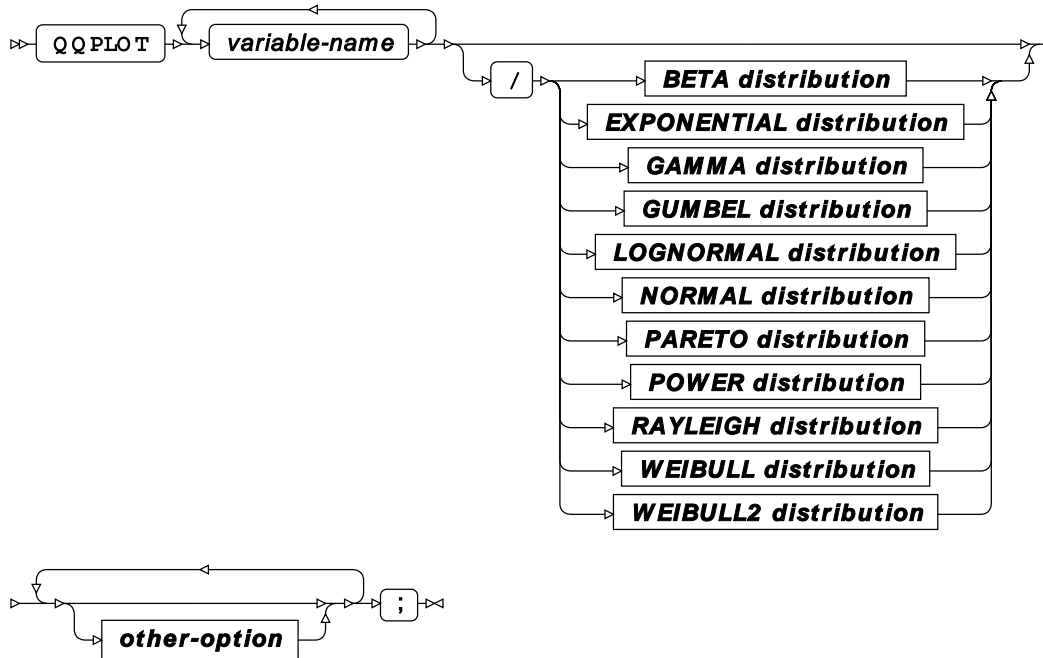


other-option

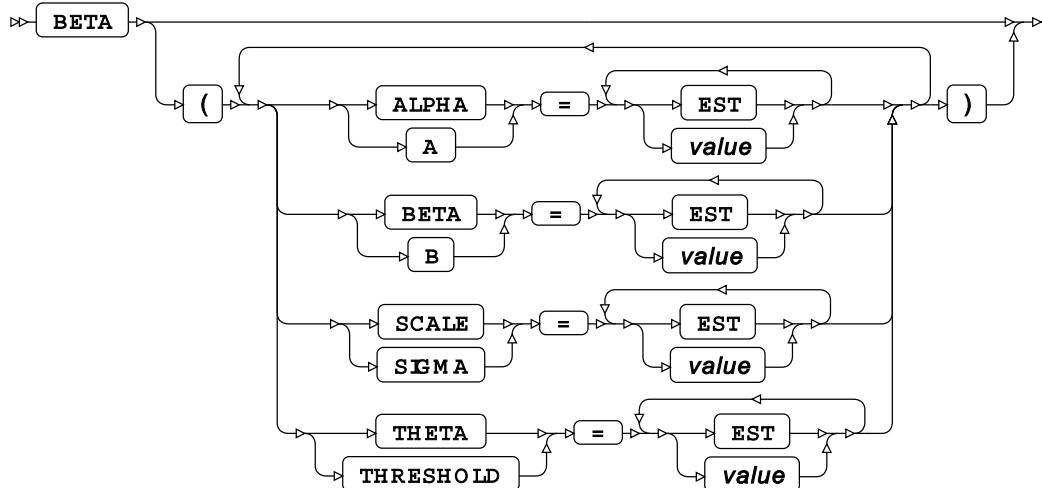


QQPLOT

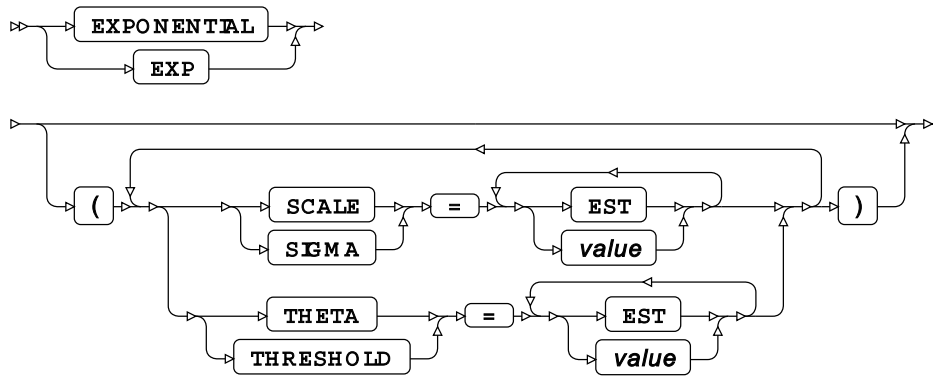
Creates quantile-quantile plots of datasets using one or more specified variables.



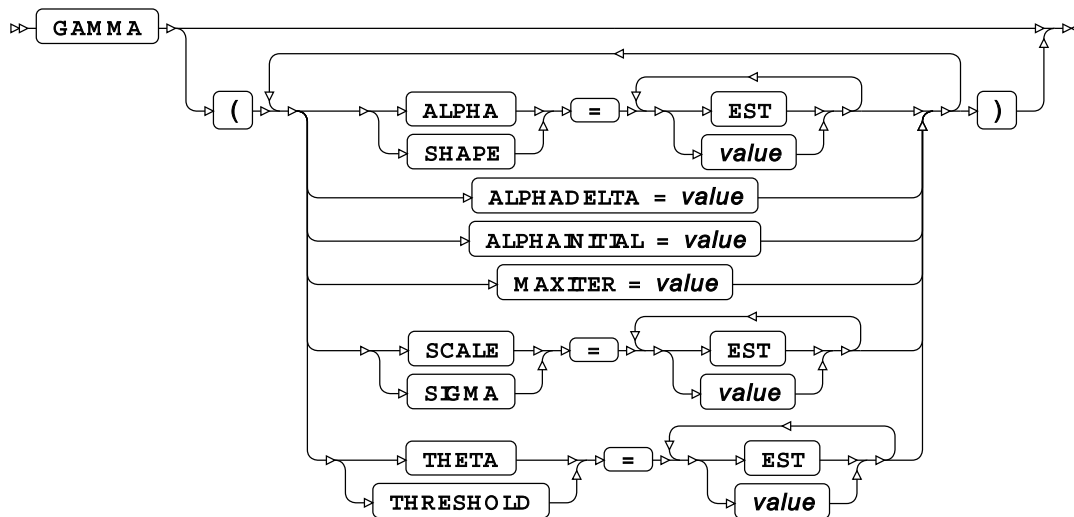
BETA distribution



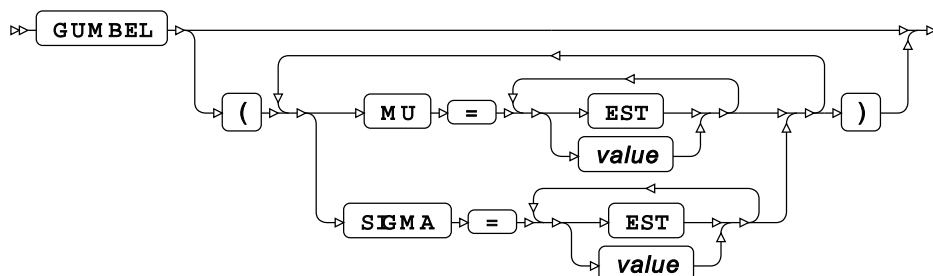
EXPONENTIAL distribution



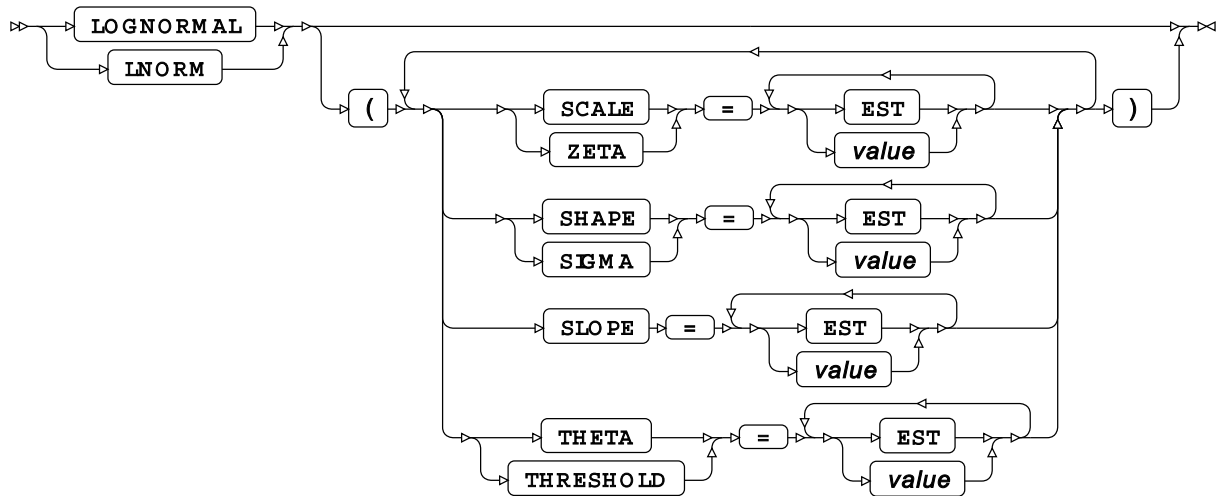
GAMMA distribution



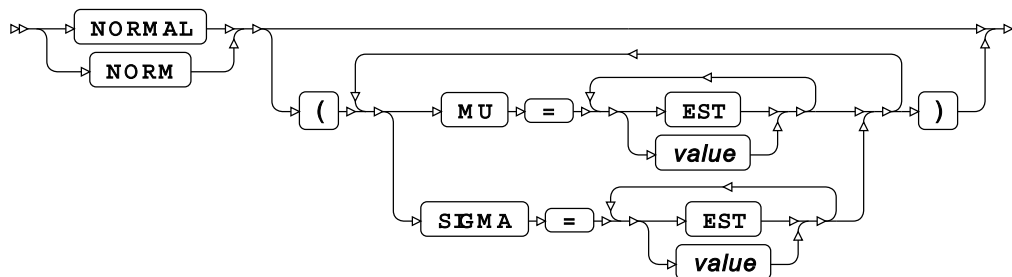
GUMBEL distribution



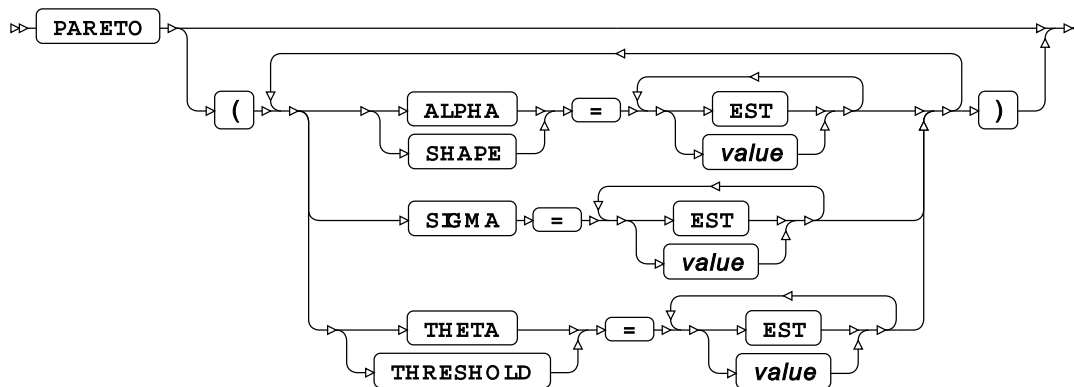
LOGNORMAL distribution



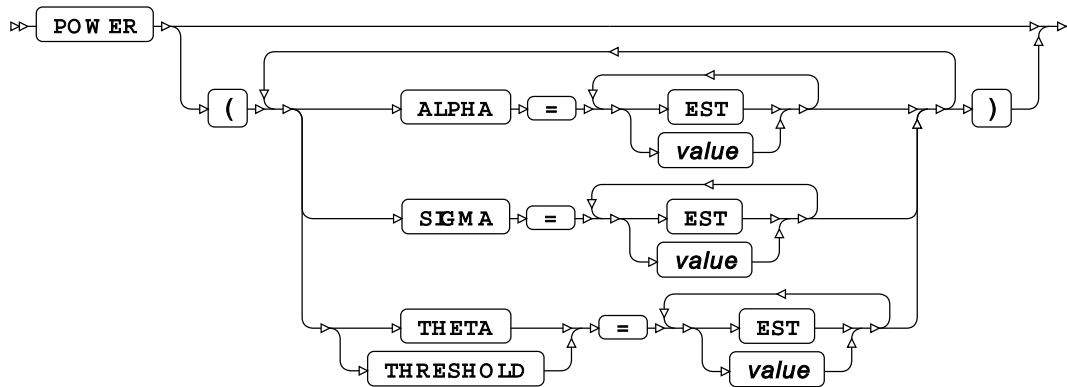
NORMAL distribution



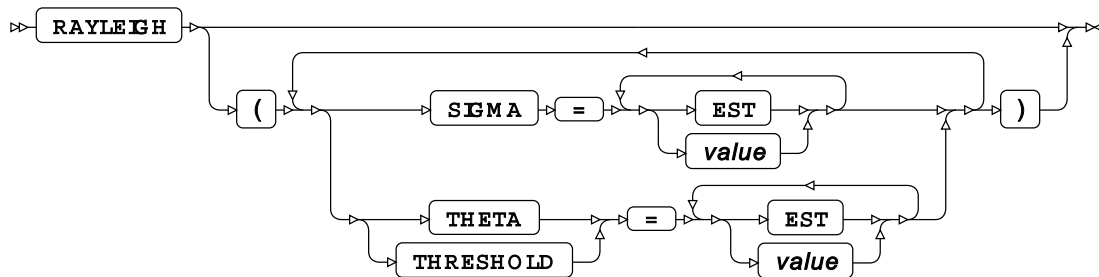
PARETO distribution



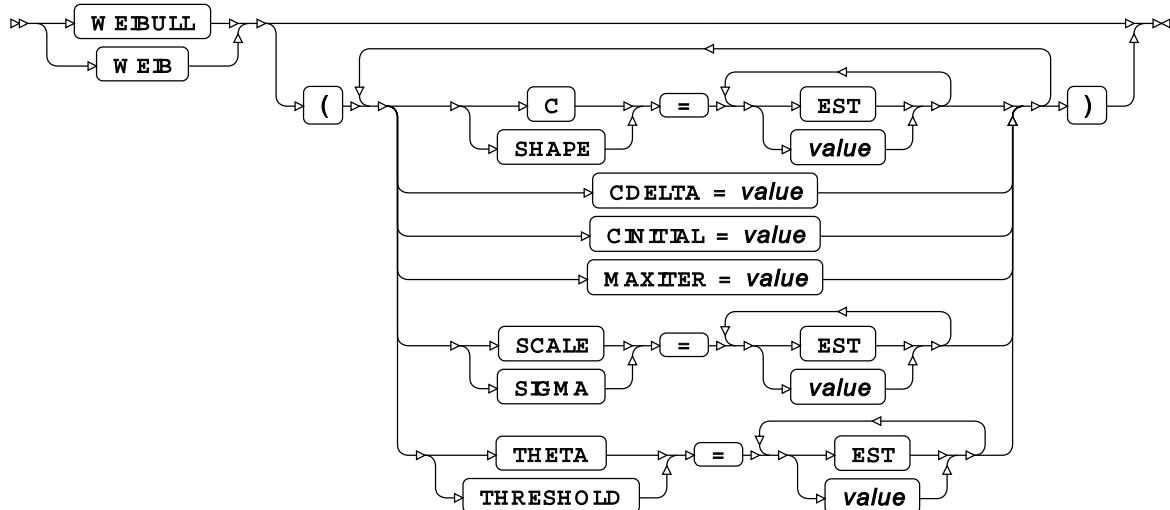
POWER distribution



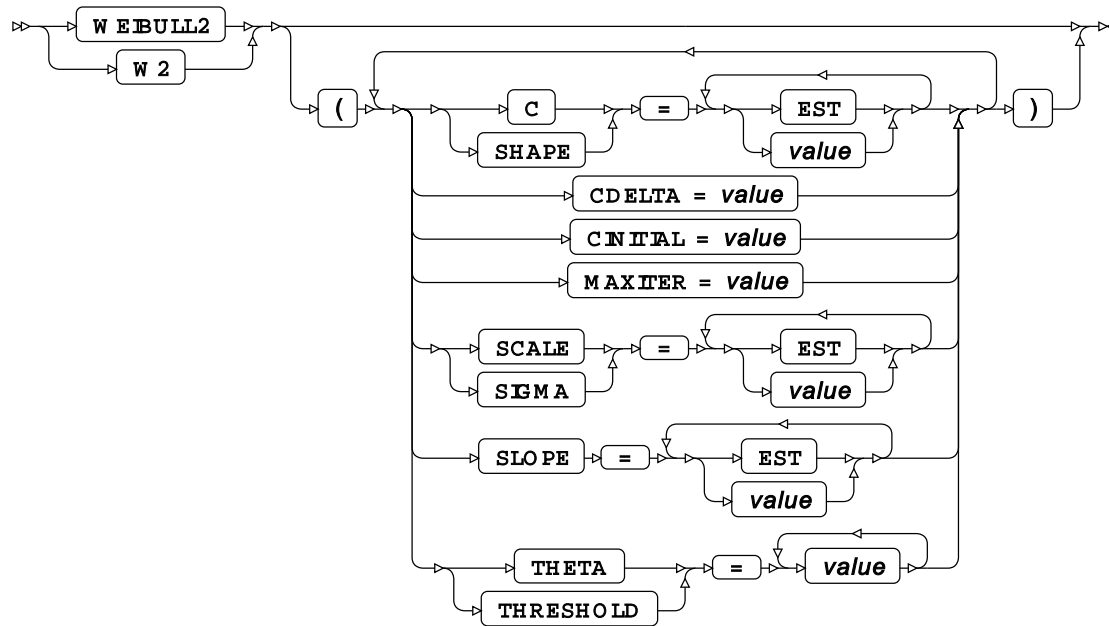
RAYLEIGH distribution



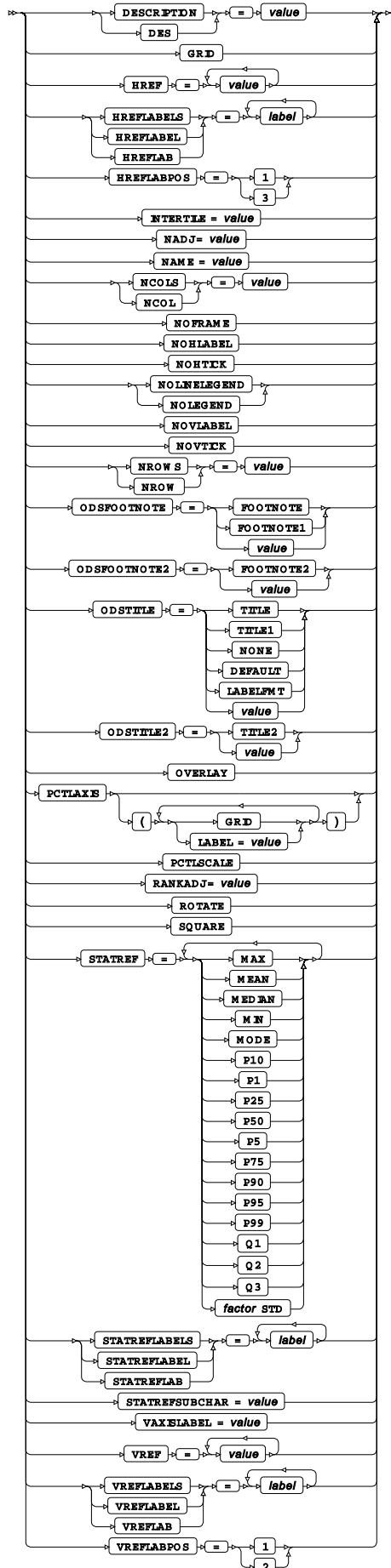
WEIBULL distribution



WEIBULL2 distribution

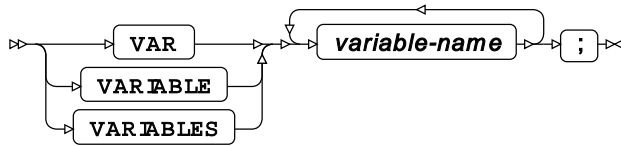


other-option



VAR

Specifies variables for which to calculate statistics and plots.



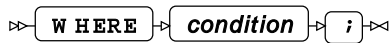
WEIGHT

Specifies a variable giving the weight associated with each observation.



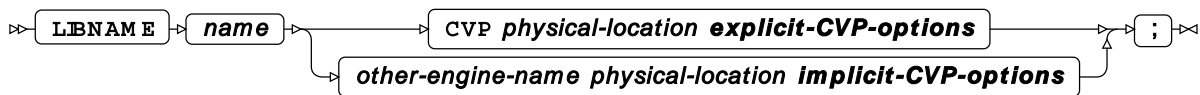
WHERE

Restricts the observations to be processed.

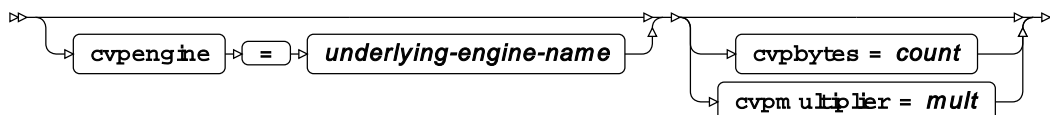


Library engines

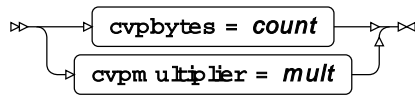
CVP



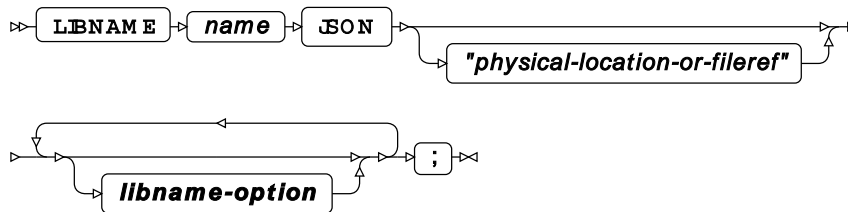
explicit-CVP-options



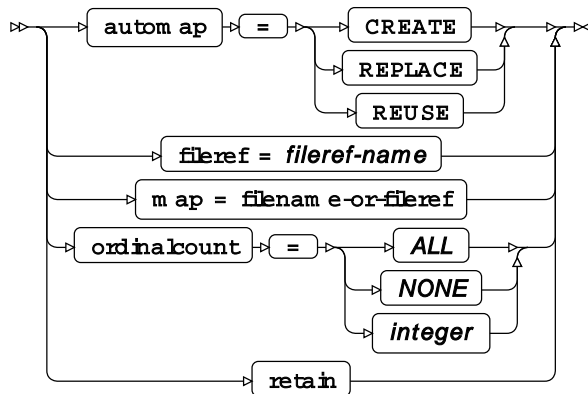
implicit-CVP-options



JSON



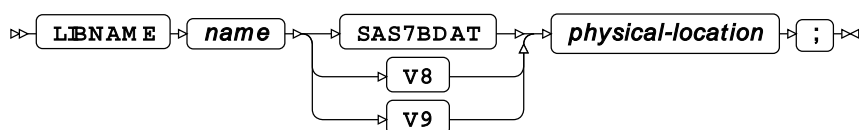
libname-option



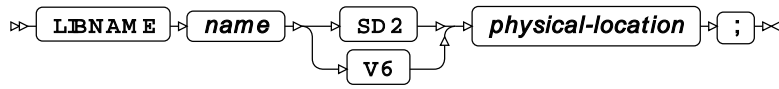
SASDASD



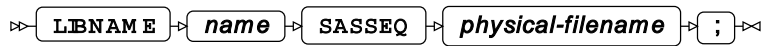
SAS7BDAT



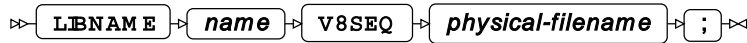
SD2



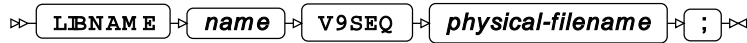
SASSEQ



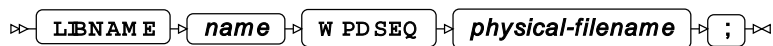
V8SEQ



V9SEQ



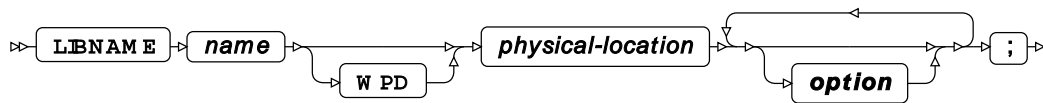
WPDSEQ



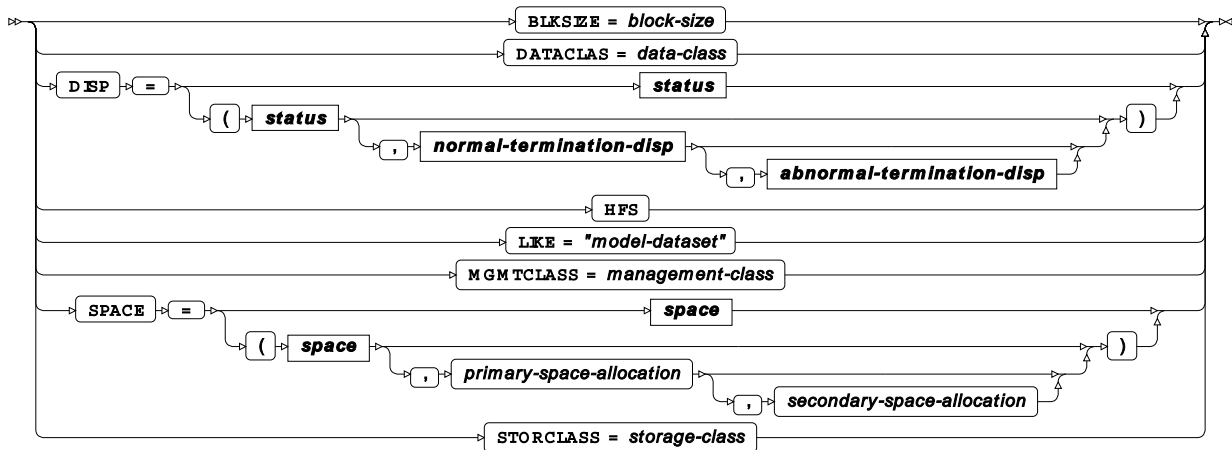
WPD



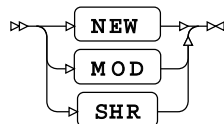
WPD (z/OS)



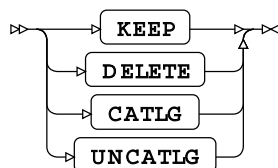
option



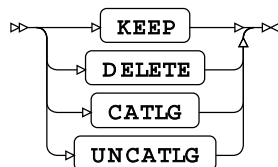
status



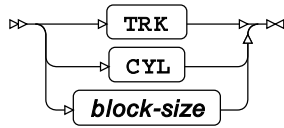
normal-termination-disp



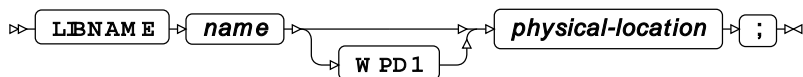
abnormal-termination-disp



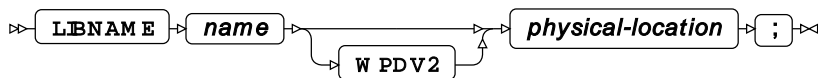
space



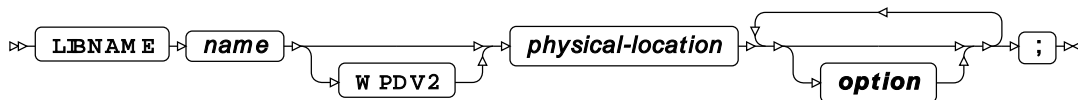
WPD1



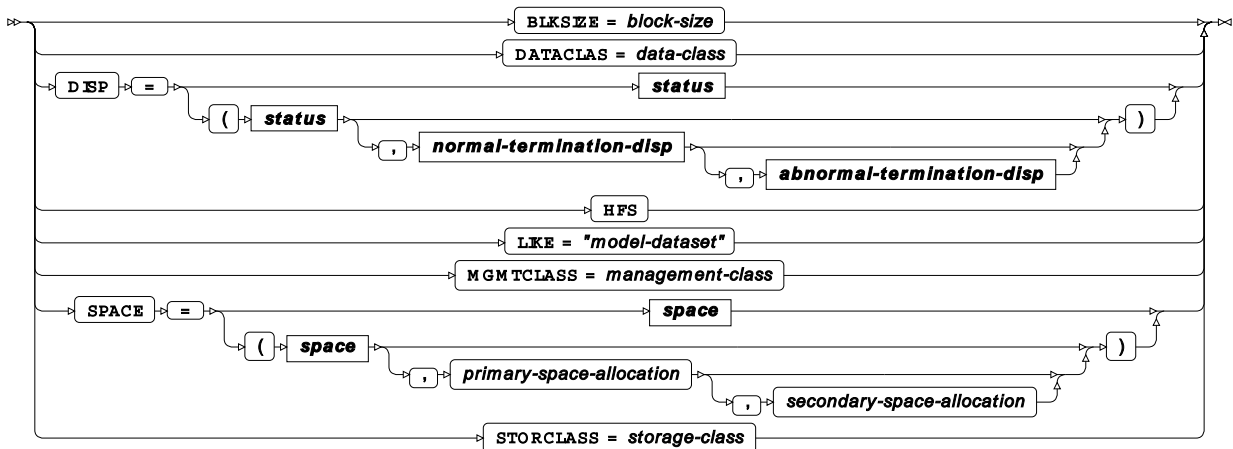
WPDV2



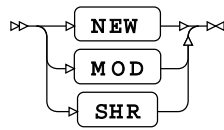
WPDV2 (z/OS)



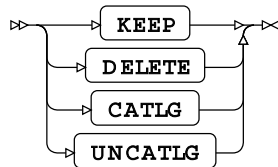
option



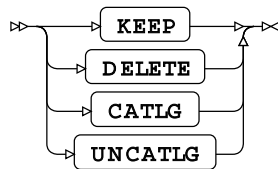
status



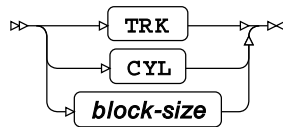
normal-termination-disp



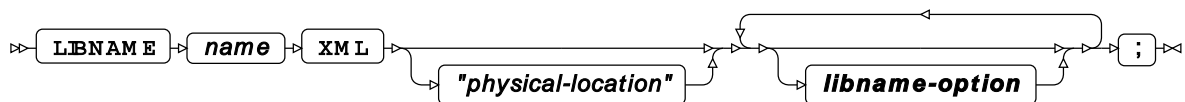
abnormal-termination-disp



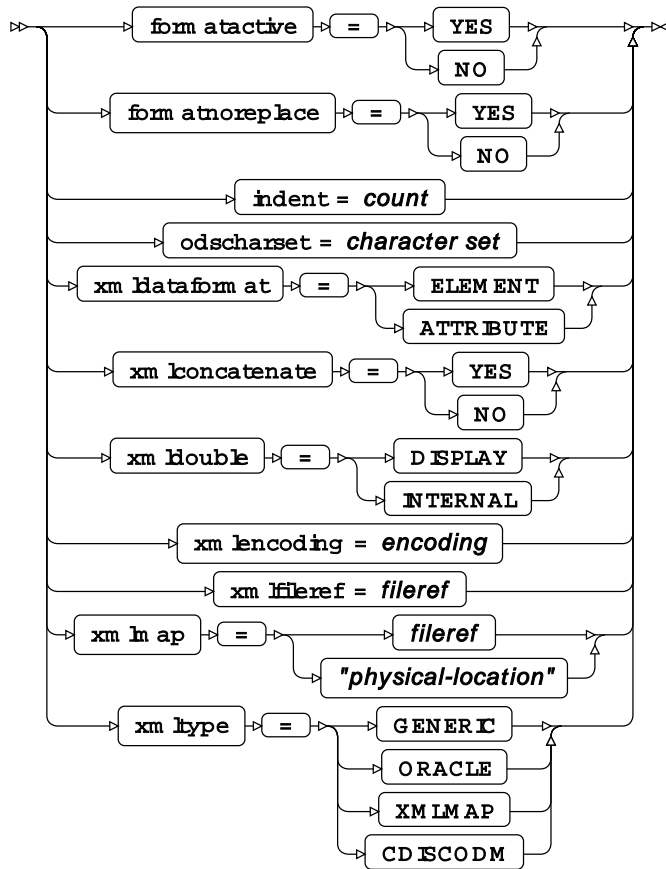
space



XML



libname-option



XPORT



Macros

Automatic macro variables

SYSADDRBITS

Read only variable.

SYSCHARWIDTH

Read only variable.

SYSCC

Read-write variable.

SYSDATE

Read only variable.

SYSDATE9

Read only variable.

SYSDAY

Read only variable.

SYSDSN

Read-write variable.

SYSENDIAN

Read only variable.

SYSENV

Read only variable.

SYSERR

Read only variable.

SYSERRORTEXT

Read-write variable.

SYSFILRC

Read-write variable.

SYSENDIAN

Read only variable.

SYSINDEX

Read only variable.

SYSINFO

Read-write variable.

SYSJOBID

Read only variable.

SYSLAST

Read-write variable.

SYSLIBRC

Read-write variable.

SYSMACRONAME

Read only variable.

SYSMAXLONG

Read only variable.

SYSTEMV

Read only variable.

SYSPARM

Read-write variable.

SYSPBUFF

Read-write variable.

SYSPROCESSID

Read only variable.

SYSPROCESSNAME

Read only variable.

SYSPROCNAME

Read only variable.

SYSRC

Read-write variable.

SYSSCP

Read-write variable.

SYSSCPL

Read-write variable.

SYSSITE

Read only variable.

SYSMAXLONG

Read only variable.

SYSsizeofPTR

Read only variable.

SYSsizeofUNICODE

Read only variable.

SYSUID

Read only variable.

SYSUSERID

Read only variable.

SYSVER

Read only variable.

SYSVLONG

SYS99ERR

Read-write variable.

SYS99INF

Read-write variable.

SYS99MSG

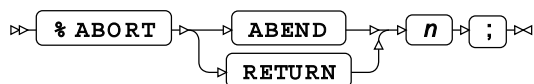
Read-write variable.

SYS99R15

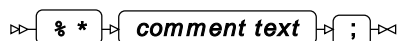
Read-write variable.

Macro processor statements

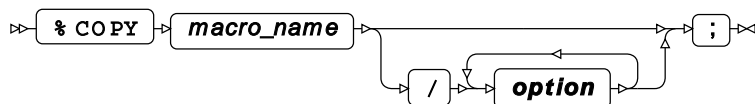
%ABORT



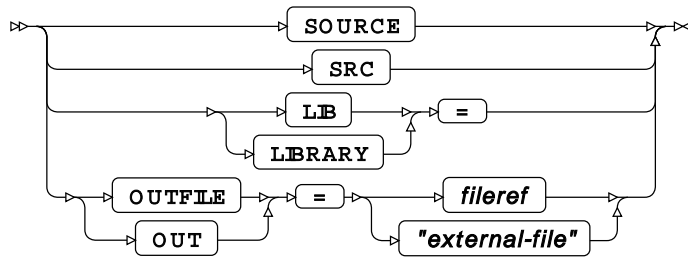
%* comment



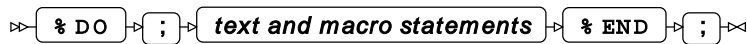
%COPY



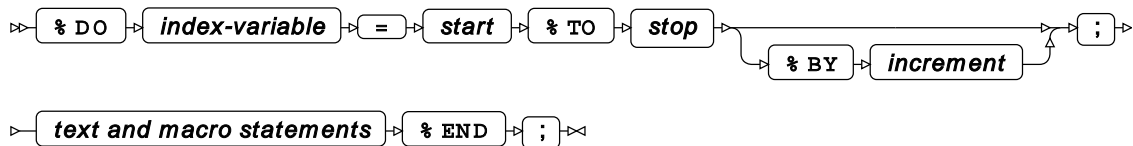
option



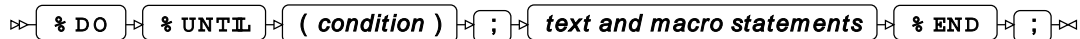
%DO



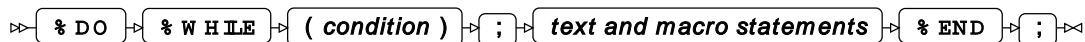
%DO, Iterative



%DO %UNTIL

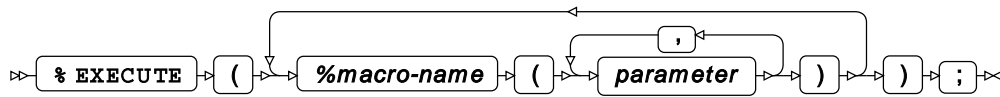


%DO %WHILE

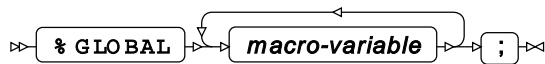


%END

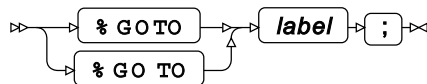
%EXECUTE



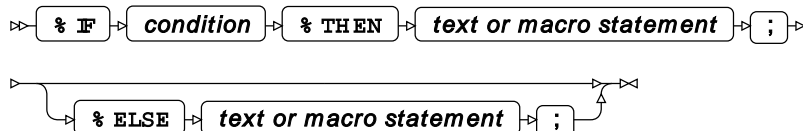
%GLOBAL



%GOTO



%IF-%THEN/%ELSE



%INCLUDE

The `%INCLUDE` statement adds source lines from an external file into the program.

For more details, please refer to the `%INCLUDE` Global Statement.

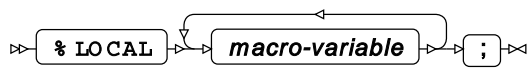
%label



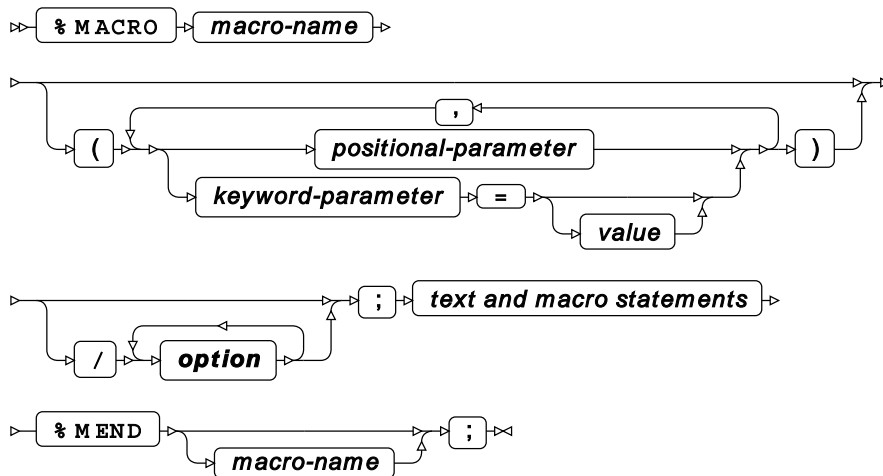
%LET



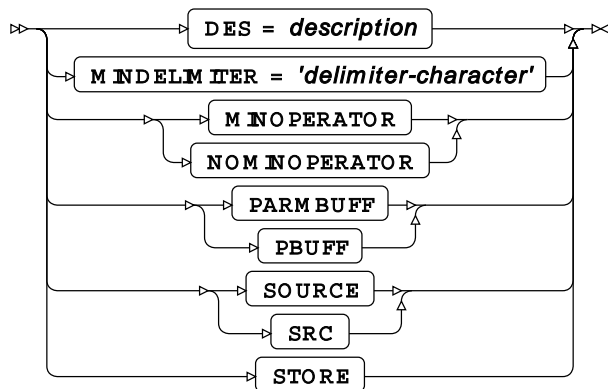
%LOCAL



%MACRO



option



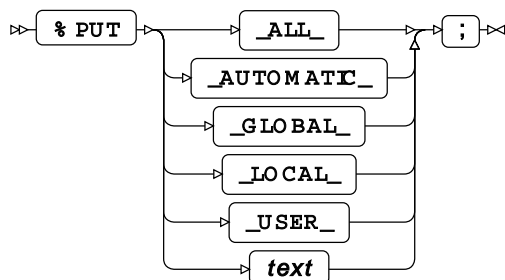
%MEND

Marks the end of the macro statement.

%MEND takes no arguments, but putting the macro-name after the statement can help in identifying the extent of a macro in your SAS language program, For example:

```
%MACRO my_macro
...
...
%MEND my_macro
```

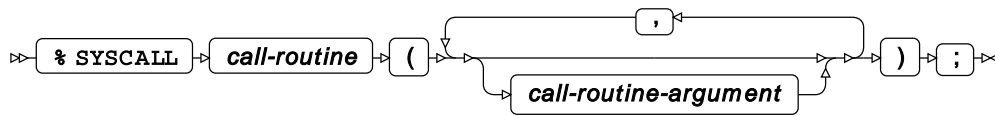
%PUT



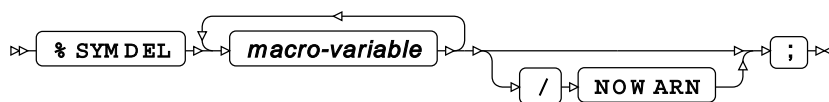
%RETURN



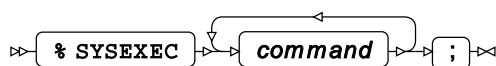
%SYSCALL



%SYMDEL

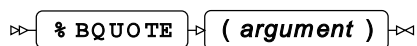


%SYSEXEC

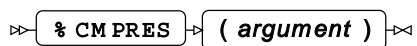


Macro processor functions

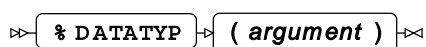
%BQUOTE



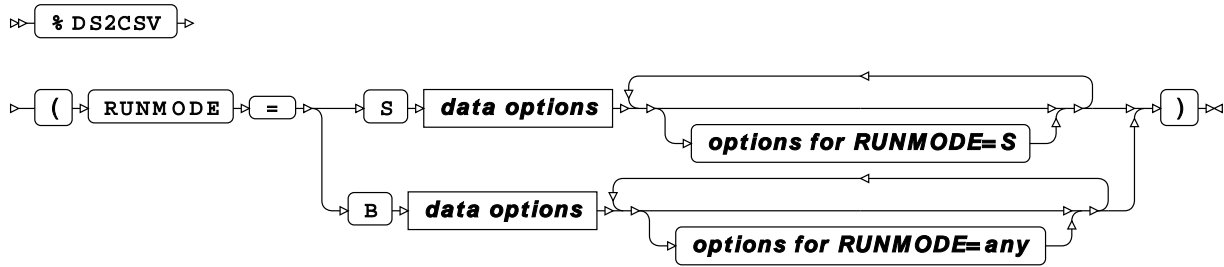
%CMPRES



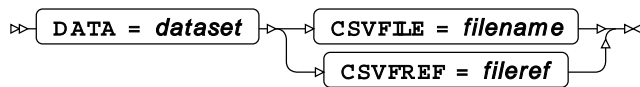
%DATATYP



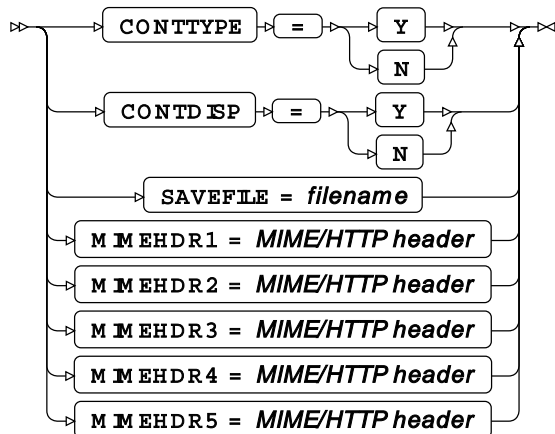
%DS2CSV



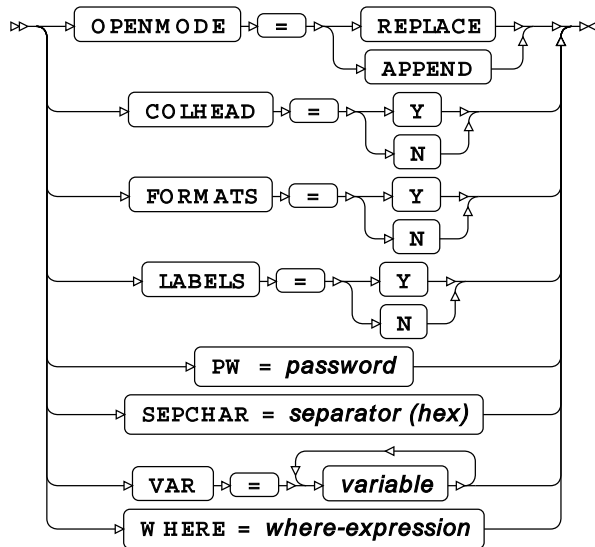
data options



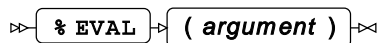
options for RUNMODE=S



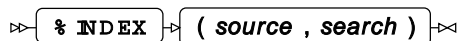
options for RUNMODE=any



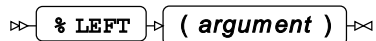
%EVAL



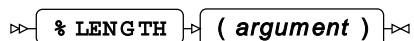
%INDEX



%LEFT



%LENGTH



%LOWCASE

⇒ % LOW CASE ⇒ (*argument*) ⇒

%NRBQUOTE

⇒ % NRBQUOTE ⇒ (*argument*) ⇒

%NRQUOTE

⇒ % NRQUOTE ⇒ (*argument*) ⇒

%NRSTR

⇒ % NRSTR ⇒ (*argument*) ⇒

%QCMPRES

⇒ % QCM PRES ⇒ (*argument*) ⇒

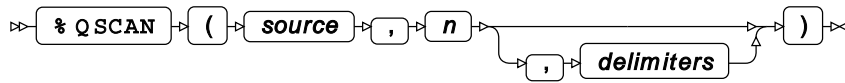
%QLEFT

⇒ % QLEFT ⇒ (*argument*) ⇒

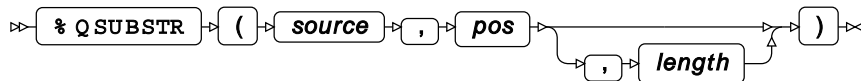
%QLOWCASE

⇒ % QLOW CASE ⇒ (*argument*) ⇒

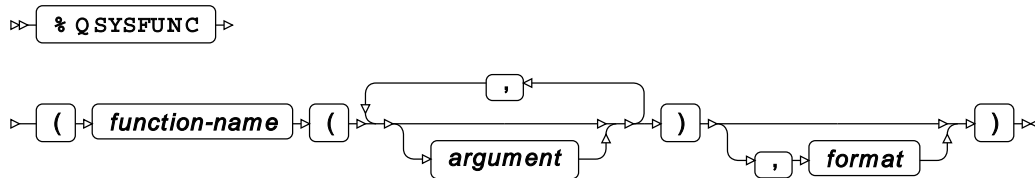
%QSCAN



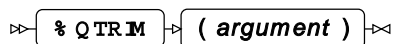
%QSUBSTR



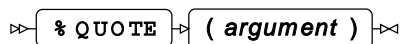
%QSYSFUNC



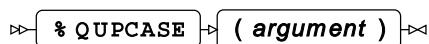
%QTRIM



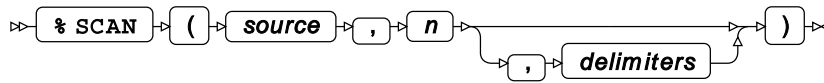
%QUOTE



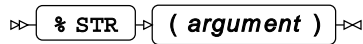
%QUPCASE



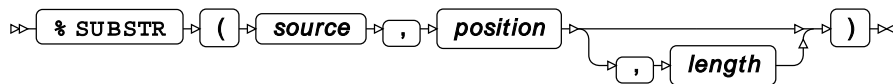
%SCAN



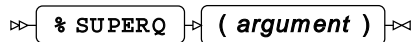
%STR



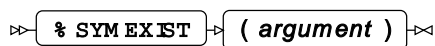
%SUBSTR



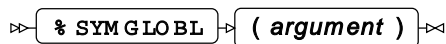
%SUPERQ



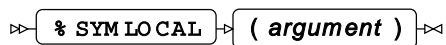
%SYMEXIST



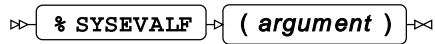
%SYMGLOBL



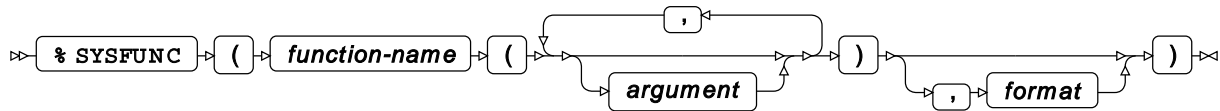
%SYMLOCAL



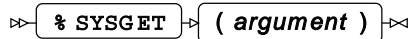
%SYSEVALF



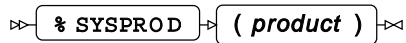
%SYSFUNC



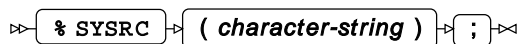
%SYSGET



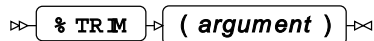
%SYSPROD



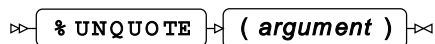
%SYSRC



%TRIM



%UNQUOTE



%UPCASE

⇒ %UPCASE ⇒ (*argument*) ⇒

%VERIFY

⇒ %VERIFY ⇒ (*source* , *excerpt*) ⇒

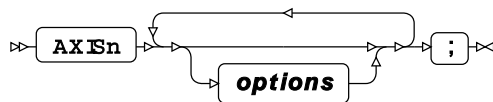
WPS Graphing

Global statements

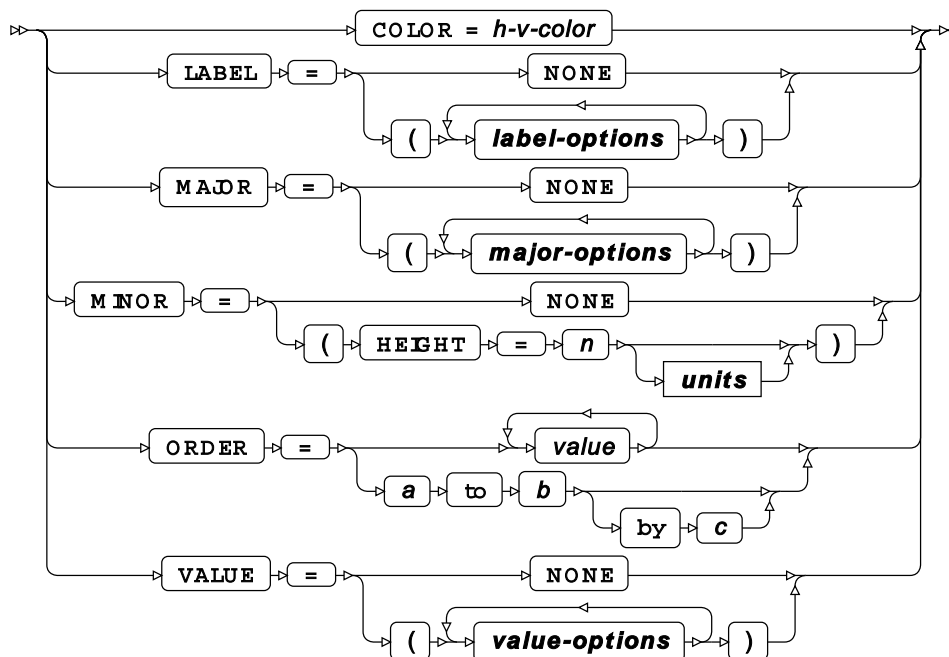
These statements create AXIS definitions, provides footnotes and graphic options, and specifies legend (key), pattern, and symbol definitions.

AXIS

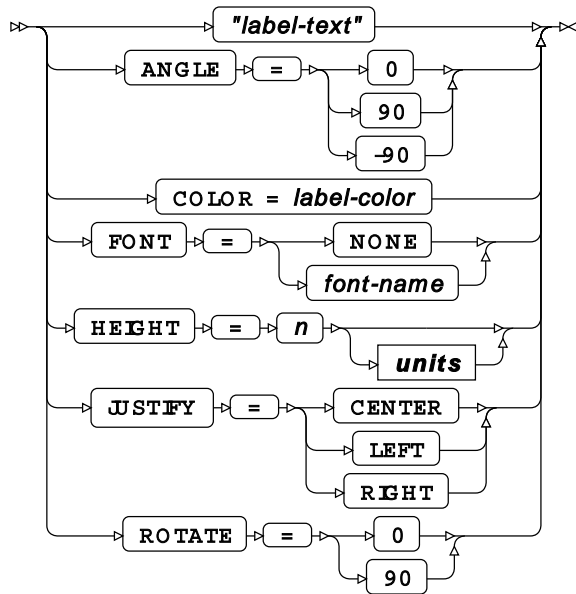
Creates AXIS definitions which control properties, such as colour.



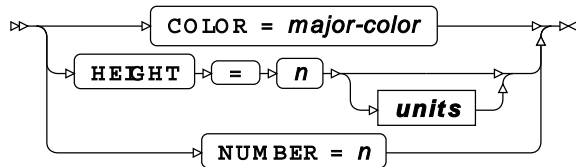
options



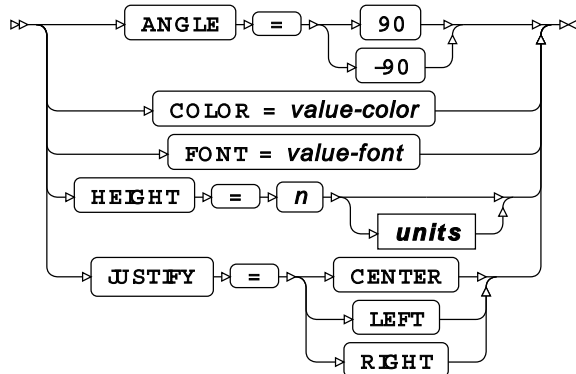
label-options



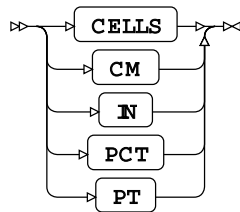
major-options



value-options

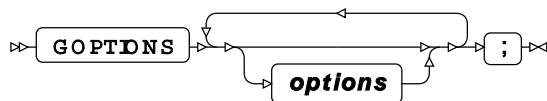


units

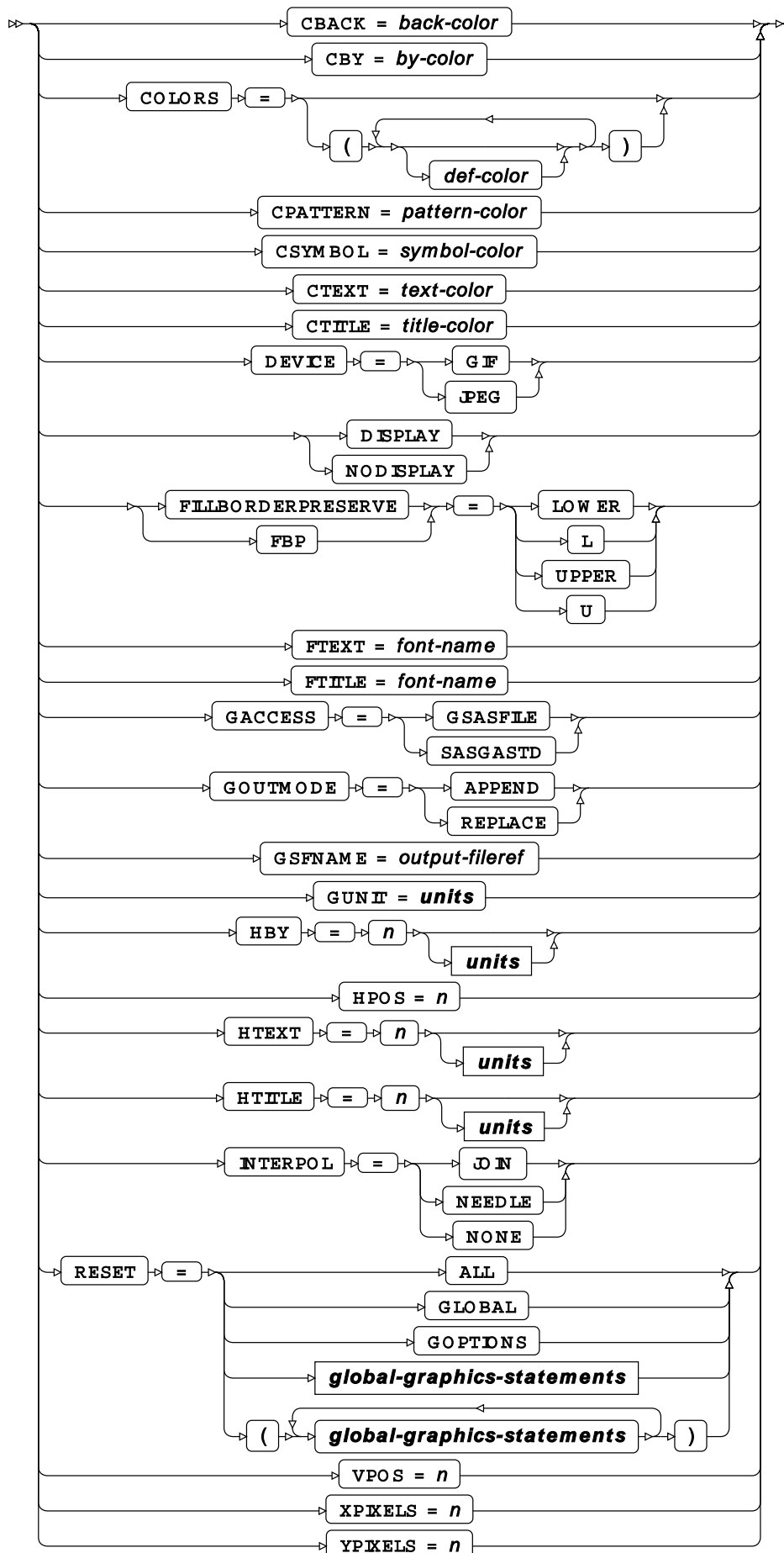


GOPTIONS

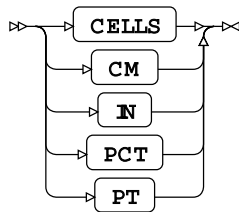
Sets various graphic options for the current session, for example, default colours, font sizes and the default device.



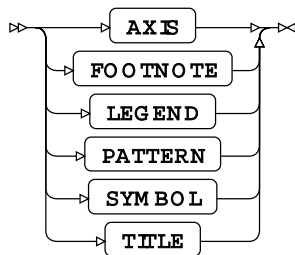
options



units

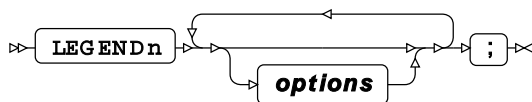


global-graphics-statements

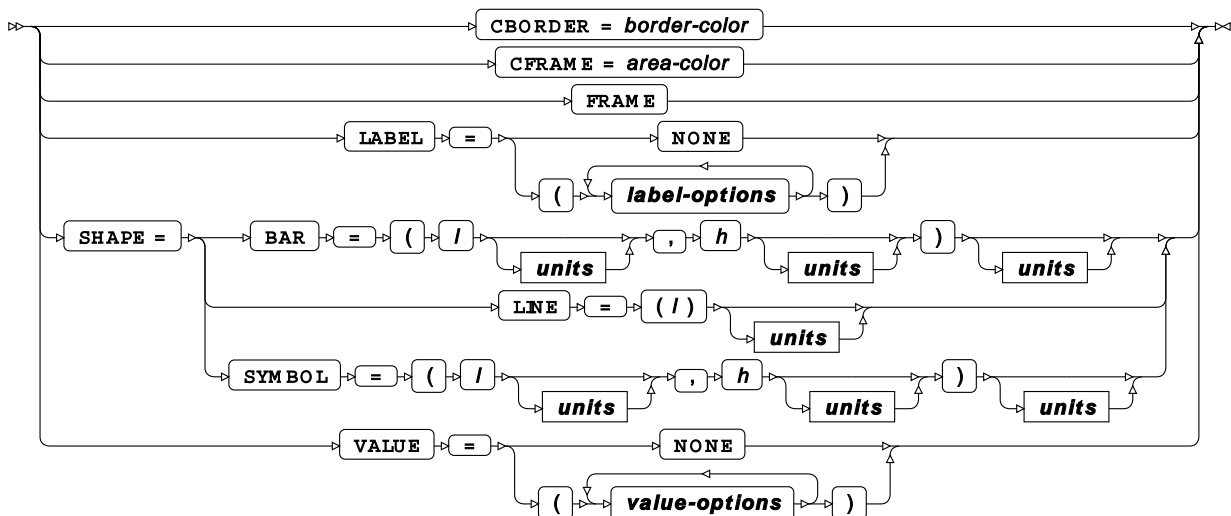


LEGEND

Specifies legend (key) definitions, for example, appearance, parameters and location.



options



The diagram illustrates a simple grammar for the sentence "The cat sat on the mat". The grammar is defined by the following rules and symbols:

- Start Symbol:** S
- Non-terminal Symbols:** S , $COLOR$, $FONT$, $HEIGHT$, n , $NONE$
- Terminal Symbols:** "label-text", "label-color", "label-font", "units"
- Rules:**
 - $S \rightarrow \text{"label-text" } COLOR = \text{label-color } FONT = NONE \text{ label-font } HEIGHT = n \text{ units}$

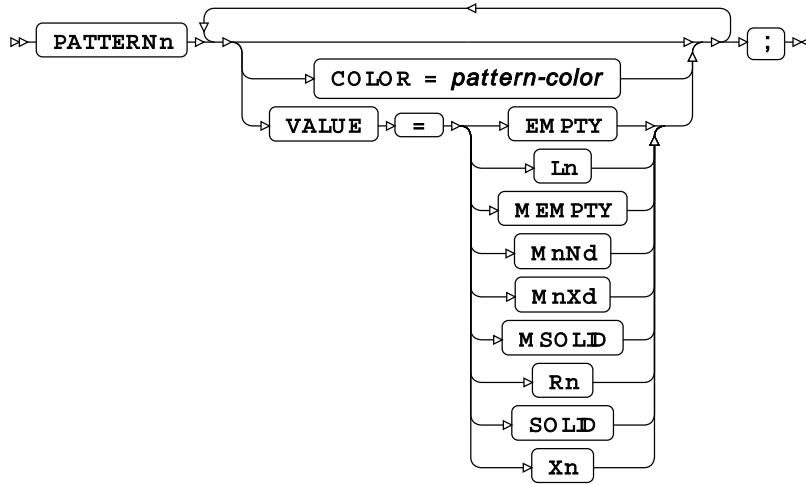
The diagram shows the derivation of the sentence from the start symbol S . The symbols "label-text", "label-color", "label-font", and "units" are terminal symbols, while S , $COLOR$, $FONT$, $HEIGHT$, n , and $NONE$ are non-terminal symbols.

```

graph LR
    Input(( )) --> J1(( ))
    J1 --> V1["value-text"]
    J1 --> J2(( ))
    J2 --> C["COLOR = value-color"]
    J2 --> J3(( ))
    J3 --> F["FONT"]
    F --> E1["="]
    E1 --> N["NONE"]
    E1 --> J4(( ))
    J4 --> VF["value-font"]
    J4 --> J5(( ))
    J5 --> H["HEIGHT"]
    H --> E2["="]
    E2 --> n["n"]
    E2 --> J6(( ))
    J6 --> U["units"]
    J6 --> J7(( ))
    J7 --> J["JUSTIFY"]
    J --> E3["="]
    E3 --> C1["CENTER"]
    E3 --> J8(( ))
    J8 --> L["LEFT"]
    J8 --> R["RIGHT"]
    J8 --> J9(( ))
    J9 --> Output(( ))
  
```

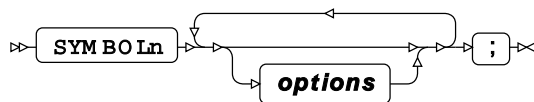
PATTERN

Specifies pattern type and colour.

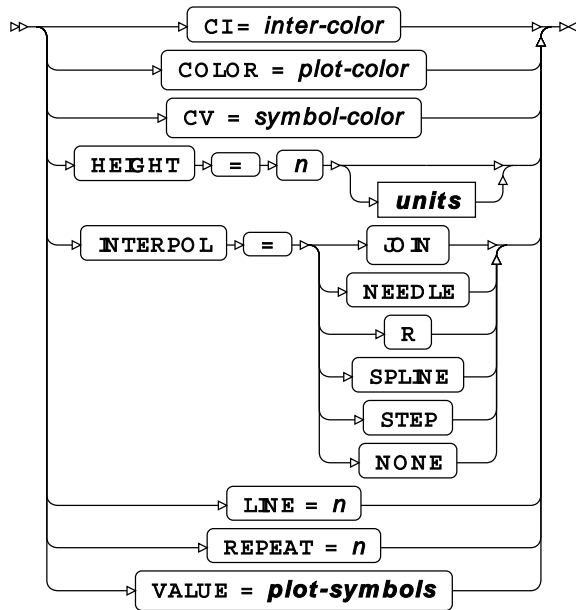


SYMBOL

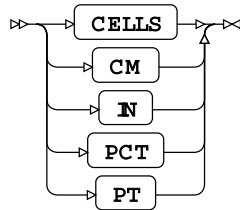
Specifies plot attributes, for example, marker size, marker colour and interpolation.



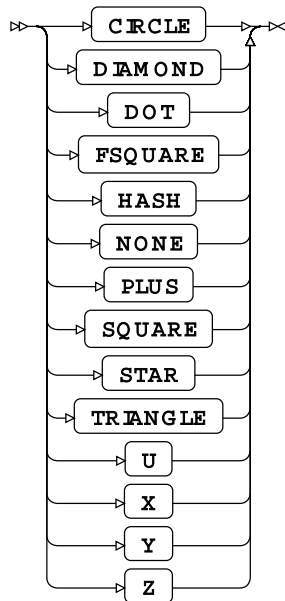
options



units



plot-symbols



Graphing procedures

These procedures create pie, horizontal and vertical displays. They also provide annotated, replay, header, and footer features.

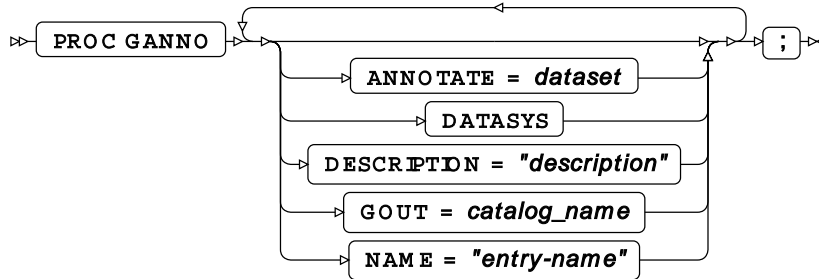
GANNO procedure

Supported statements

- `PROC GANNO` [↗](#) (page 2613)

PROC GANNO

Creates graphical output from instructions stored in an ANNOTATE dataset.



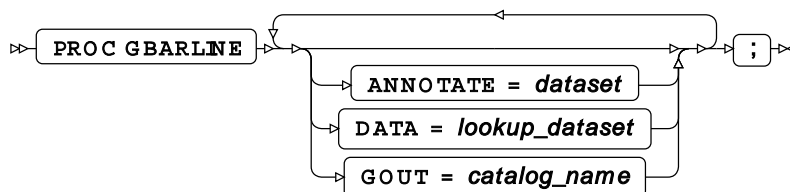
GBARLINE procedure

Supported statements

- *PROC GBARLINE* [↗](#) (page 2613)
- *BAR* [↗](#) (page 2614)
- *BY* [↗](#) (page 2616)
- *FORMAT* [↗](#) (page 2616)
- *LABEL* [↗](#) (page 2616)
- *PLOT* [↗](#) (page 2616)
- *WHERE* [↗](#) (page 2617)

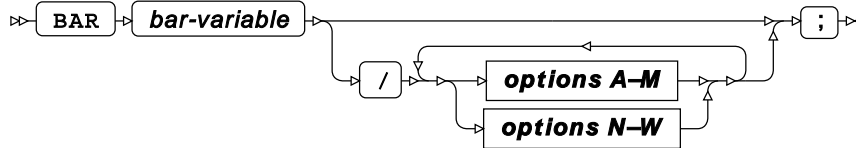
PROC GBARLINE

Creates a vertical bar chart using specified data. Also allows plot data to be overlaid.

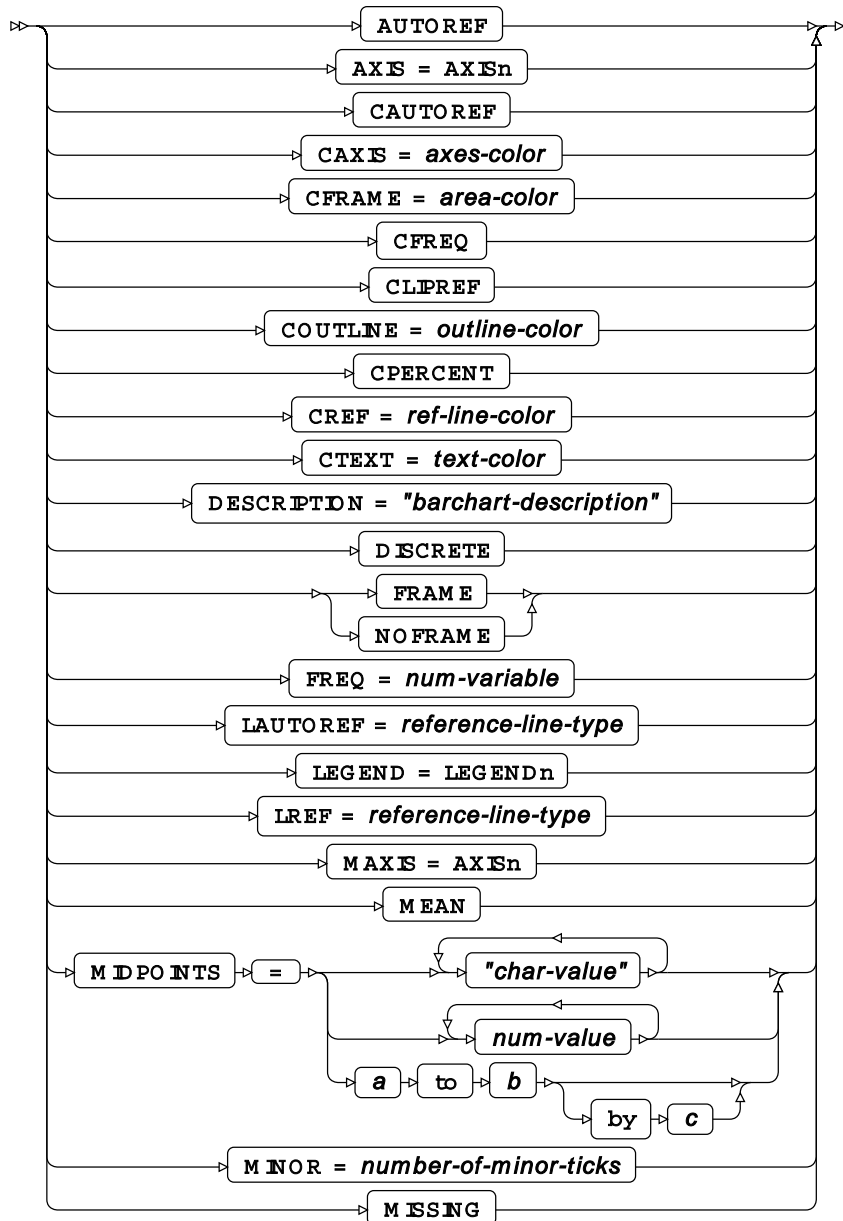


BAR

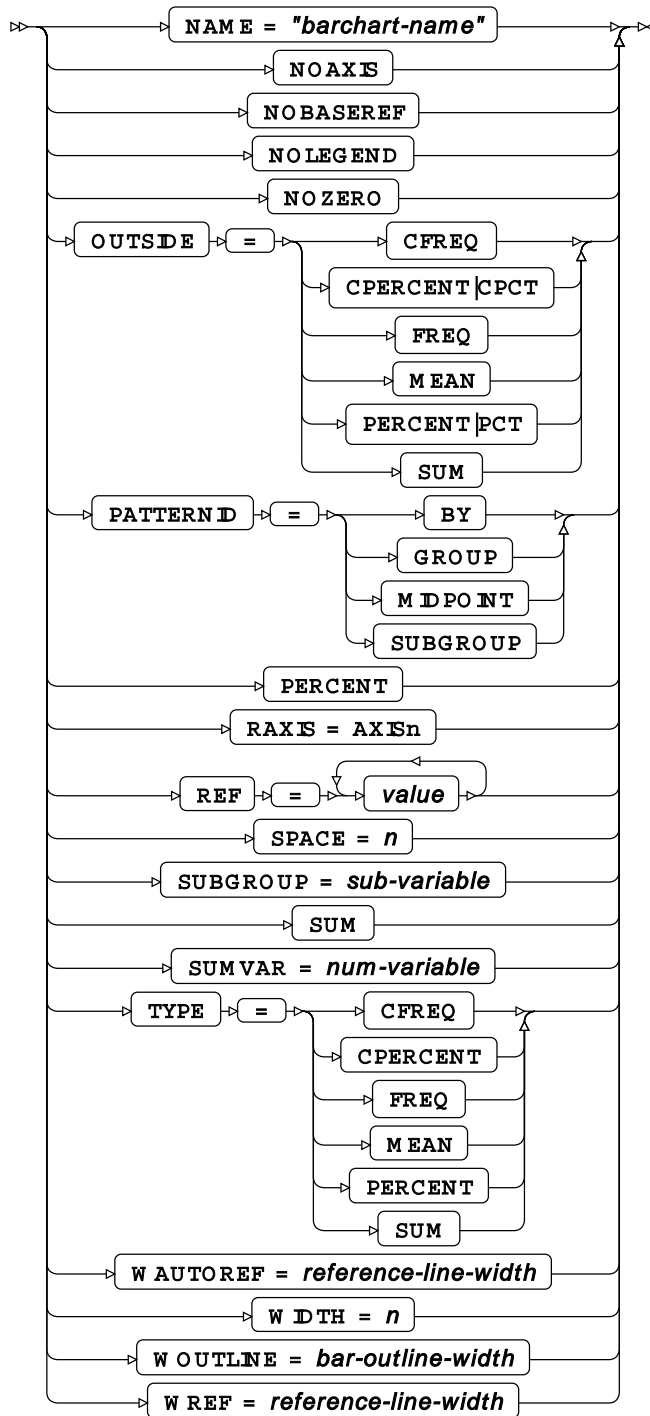
Creates vertical bar charts.



options A–M

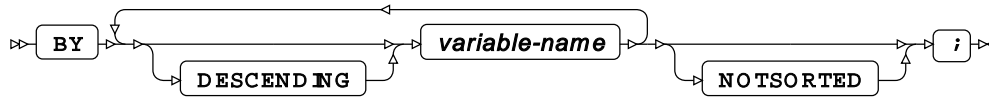


options N–W



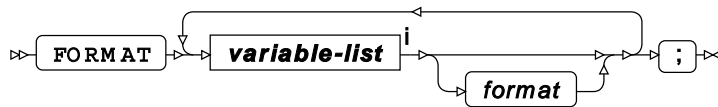
BY

Groups the observations in a dataset using one or more specified variables.



FORMAT

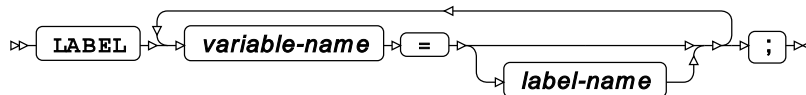
Adds formats to one or more variables in a dataset.



ⁱ See [Variable Lists](#) (page 32).

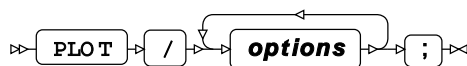
LABEL

Adds labels to one or more variables in a dataset.

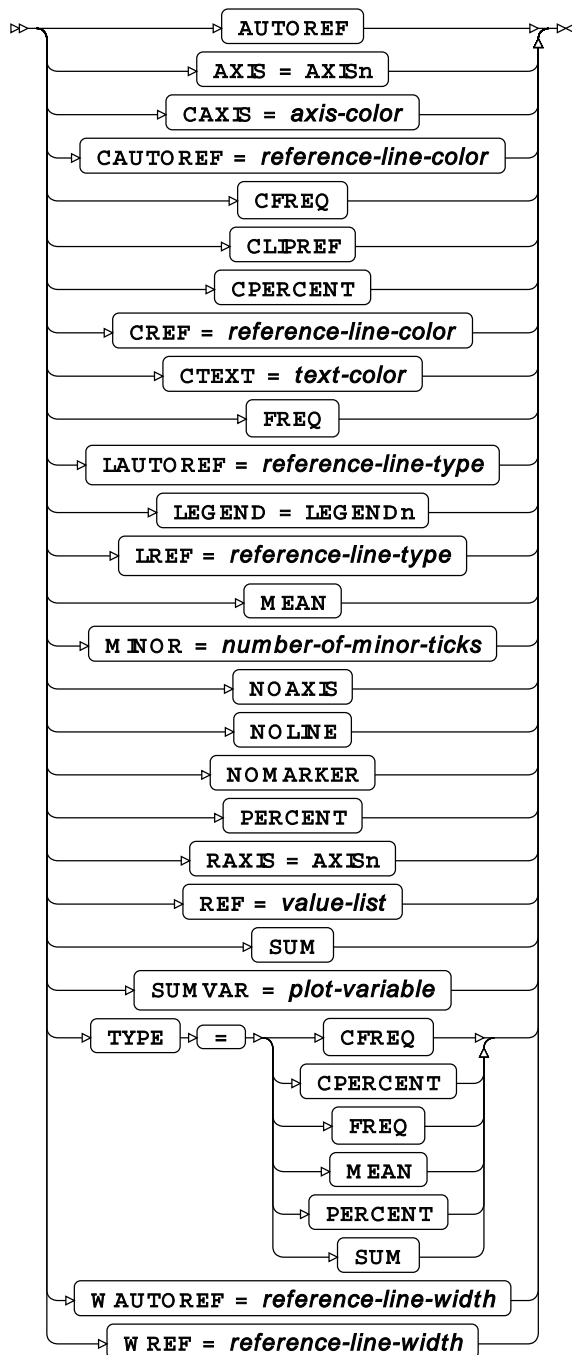


PLOT

Overlays a plot on the bar chart. The variables are plotted on the horizontal and right-hand side.

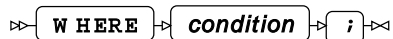


options



WHERE

Restricts the observations to be processed.



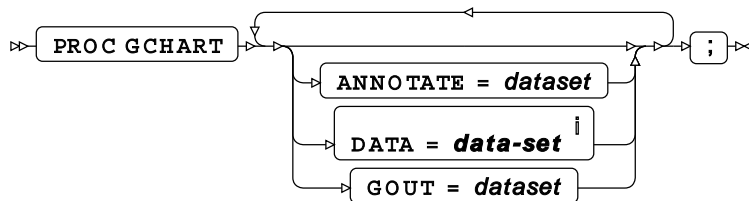
GCHART procedure

Supported statements

- *PROC GCHART* [↗](#) (page 2618)
- *BY* [↗](#) (page 2618)
- *FORMAT* [↗](#) (page 2619)
- *HBAR* [↗](#) (page 2619)
- *LABEL* [↗](#) (page 2622)
- *PIE* [↗](#) (page 2622)
- *VBAR* [↗](#) (page 2624)
- *WHERE* [↗](#) (page 2626)

PROC GCHART

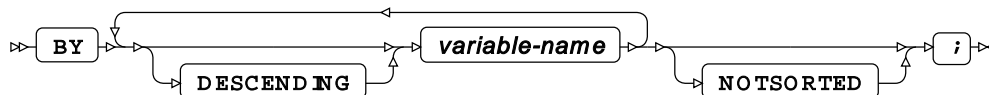
Enables the creation of multiple horizontal and vertical bar charts, and pie charts.



ⁱ See *Input dataset* [↗](#) (page 16).

BY

Groups the observations in a dataset using one or more specified variables.



FORMAT

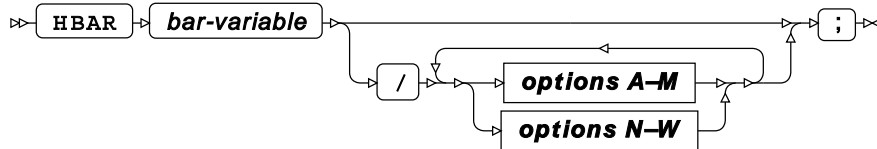
Adds formats to one or more variables in a dataset.



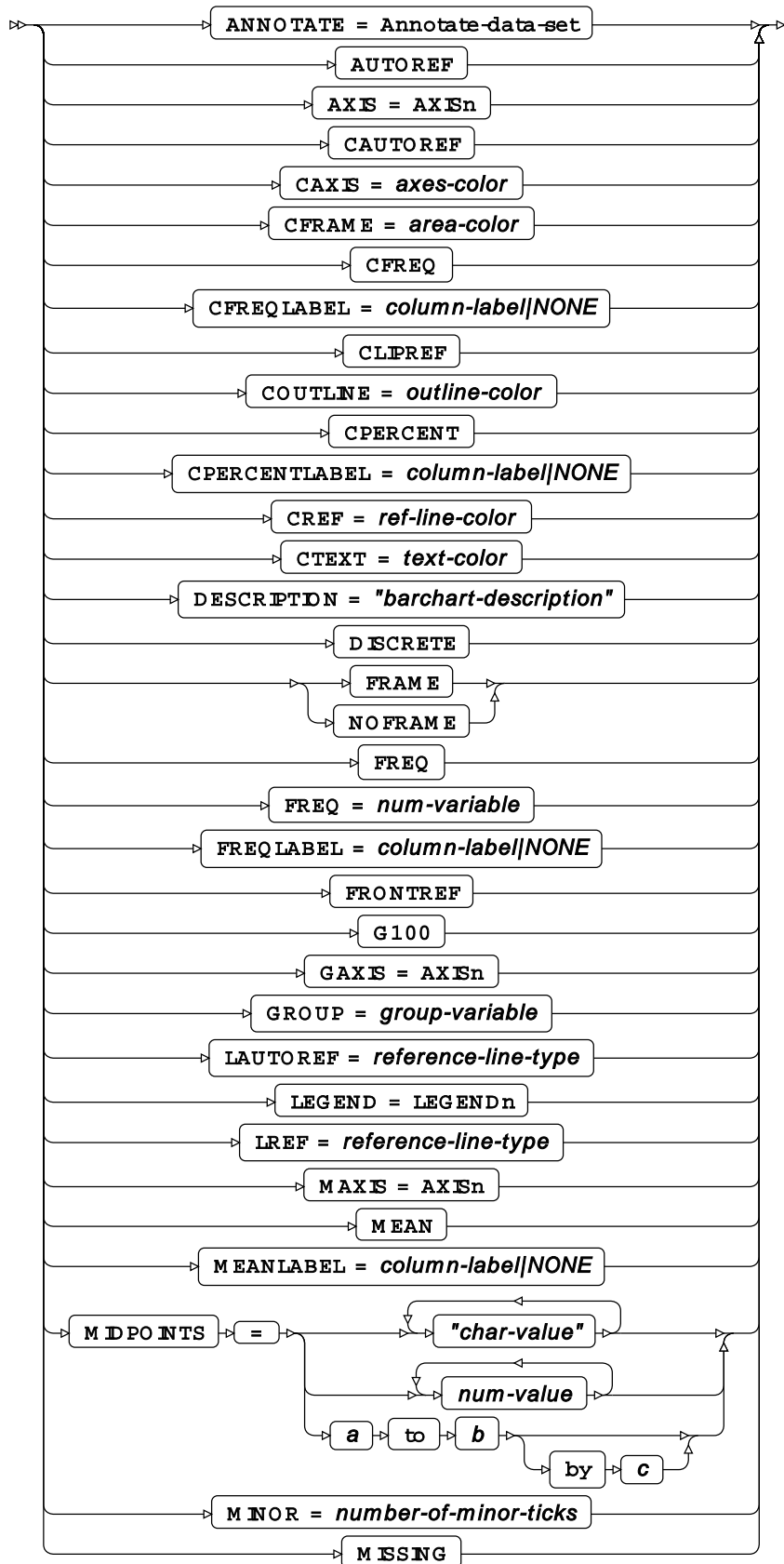
ⁱ See *Variable Lists* [↗](#) (page 32).

HBAR

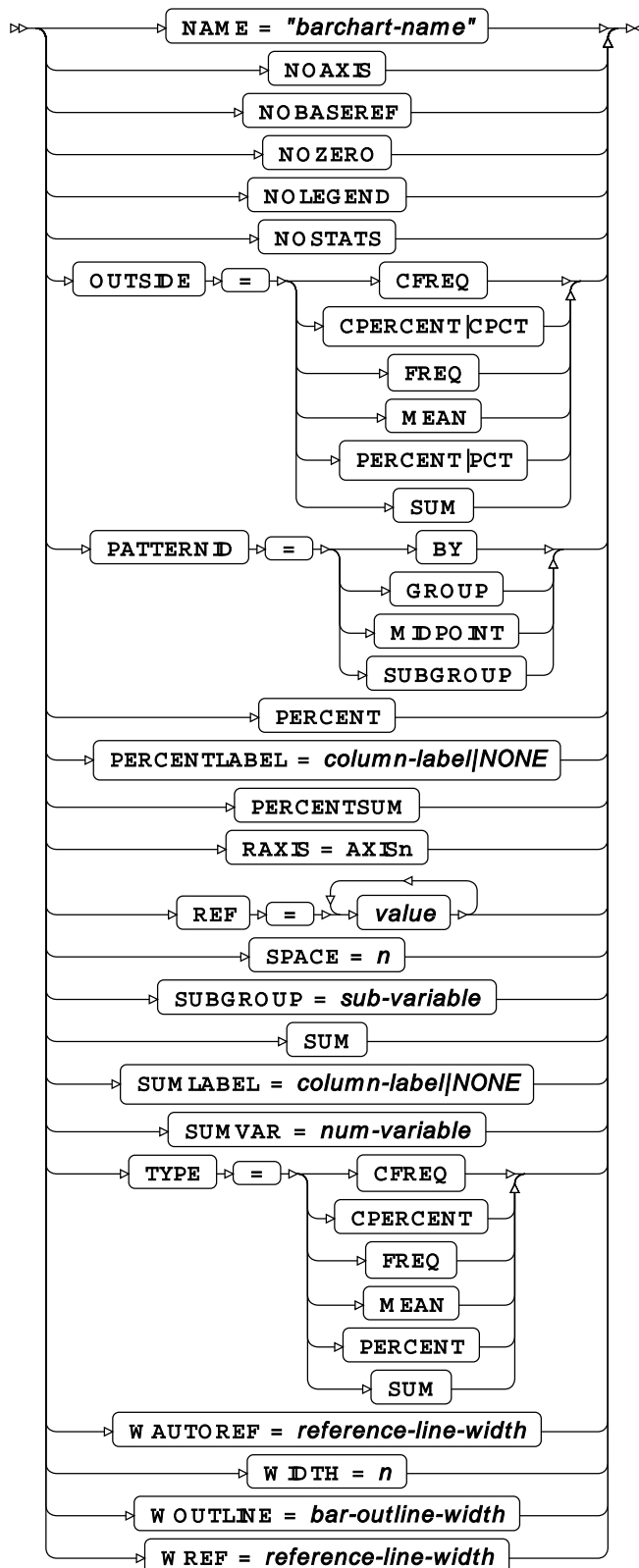
Creates horizontal bar charts.



options A–M

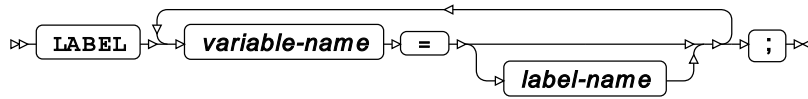


options N–W



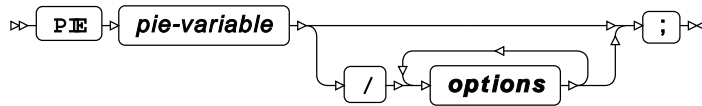
LABEL

Adds labels to one or more variables in a dataset.

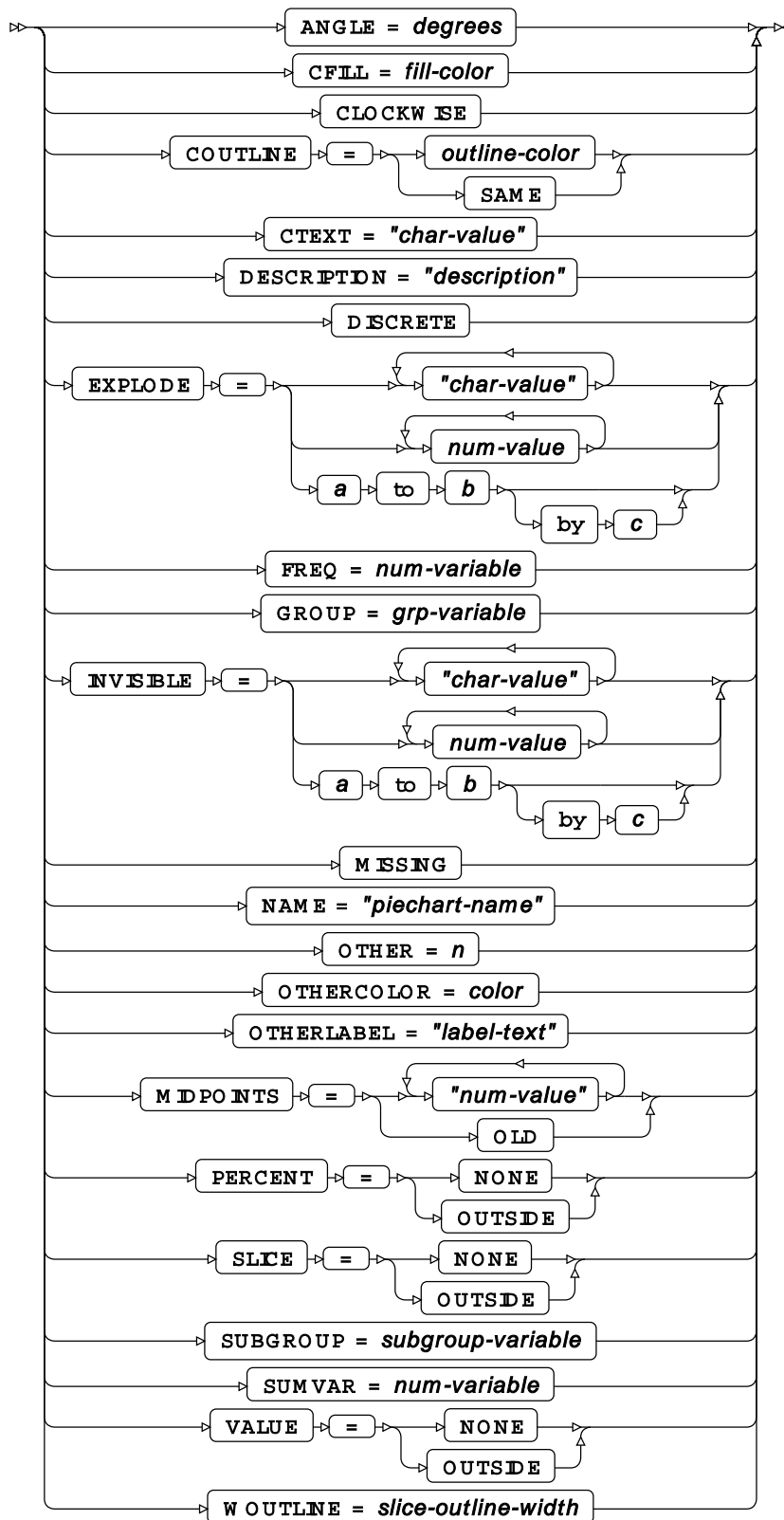


PIE

Creates pie charts.

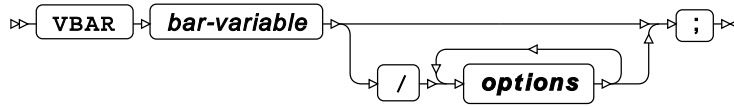


options

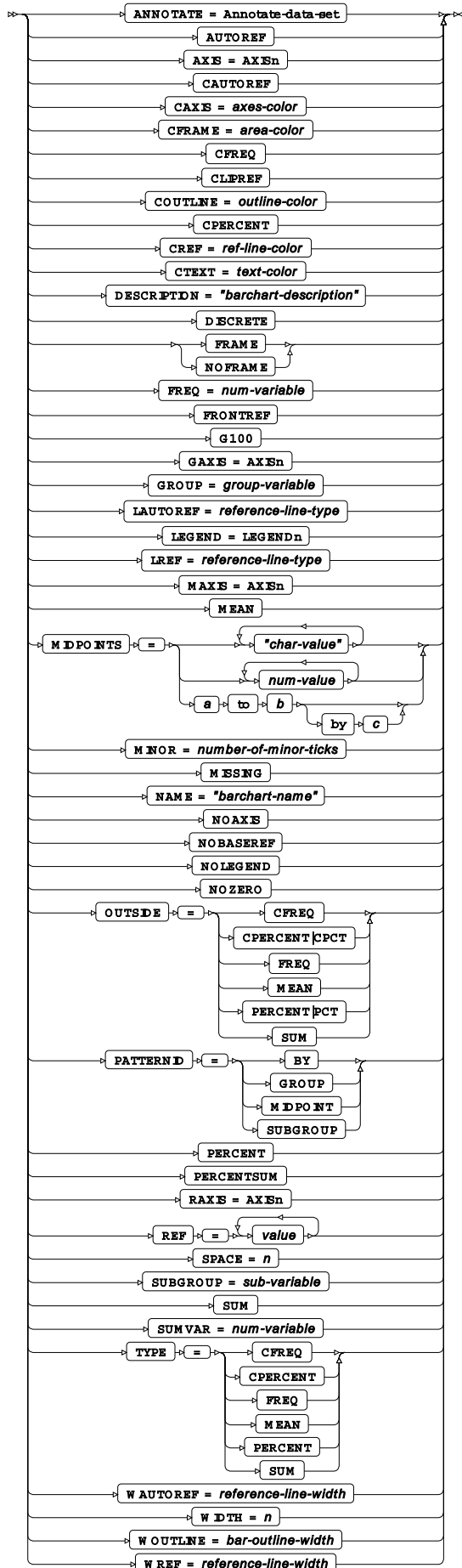


VBAR

Creates vertical bar charts.

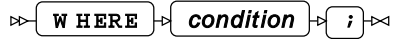


options



WHERE

Restricts the observations to be processed.



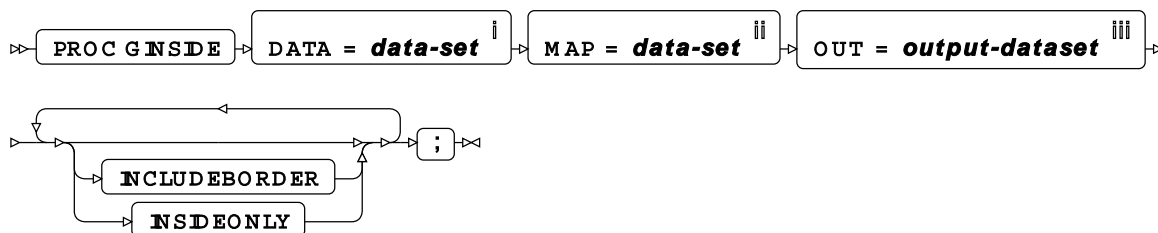
GINSIDE procedure

Supported statements

- *PROC GINSIDE* [↗](#) (page 2626)
- *ID* [↗](#) (page 2626)

PROC GINSIDE

Tests a point to determine whether it is inside a polygon.



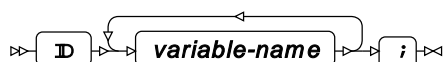
ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Output dataset* [↗](#) (page 16).

ID

Specifies the variables in a map dataset that define map areas.



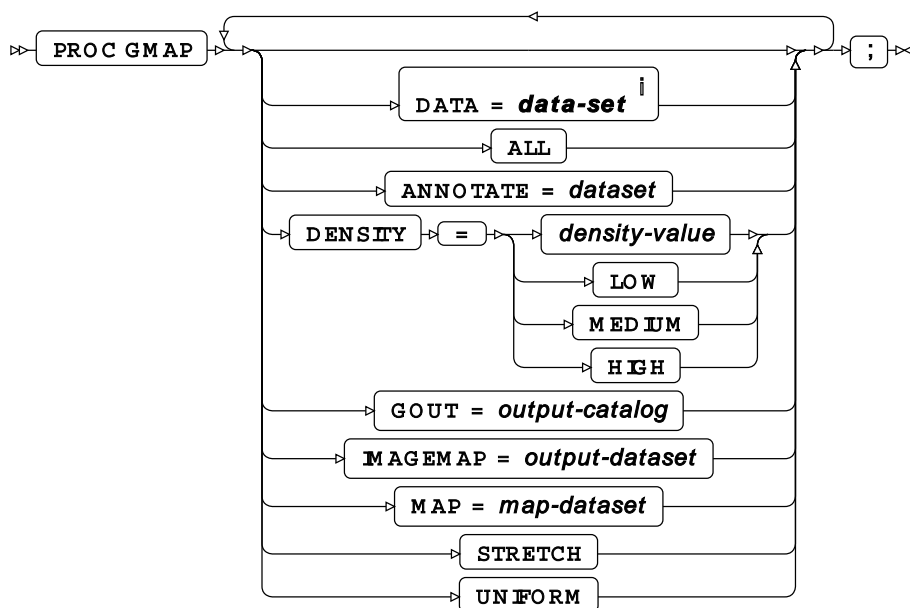
GMAP procedure

Supported statements

- *PROC GMAP* [↗](#) (page 2627)
- *BY* [↗](#) (page 2628)
- *CHORO* [↗](#) (page 2628)
- *FORMAT* [↗](#) (page 2630)
- *ID* [↗](#) (page 2630)
- *LABEL* [↗](#) (page 2630)
- *WHERE* [↗](#) (page 2630)

PROC GMAP

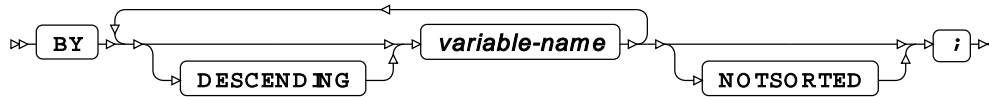
Creates a region-coloured map from a specified dataset containing map data points.



ⁱ See *Input dataset* [↗](#) (page 16).

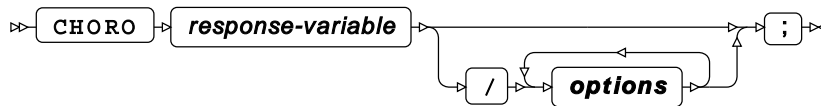
BY

Groups the observations in a dataset using one or more specified variables.

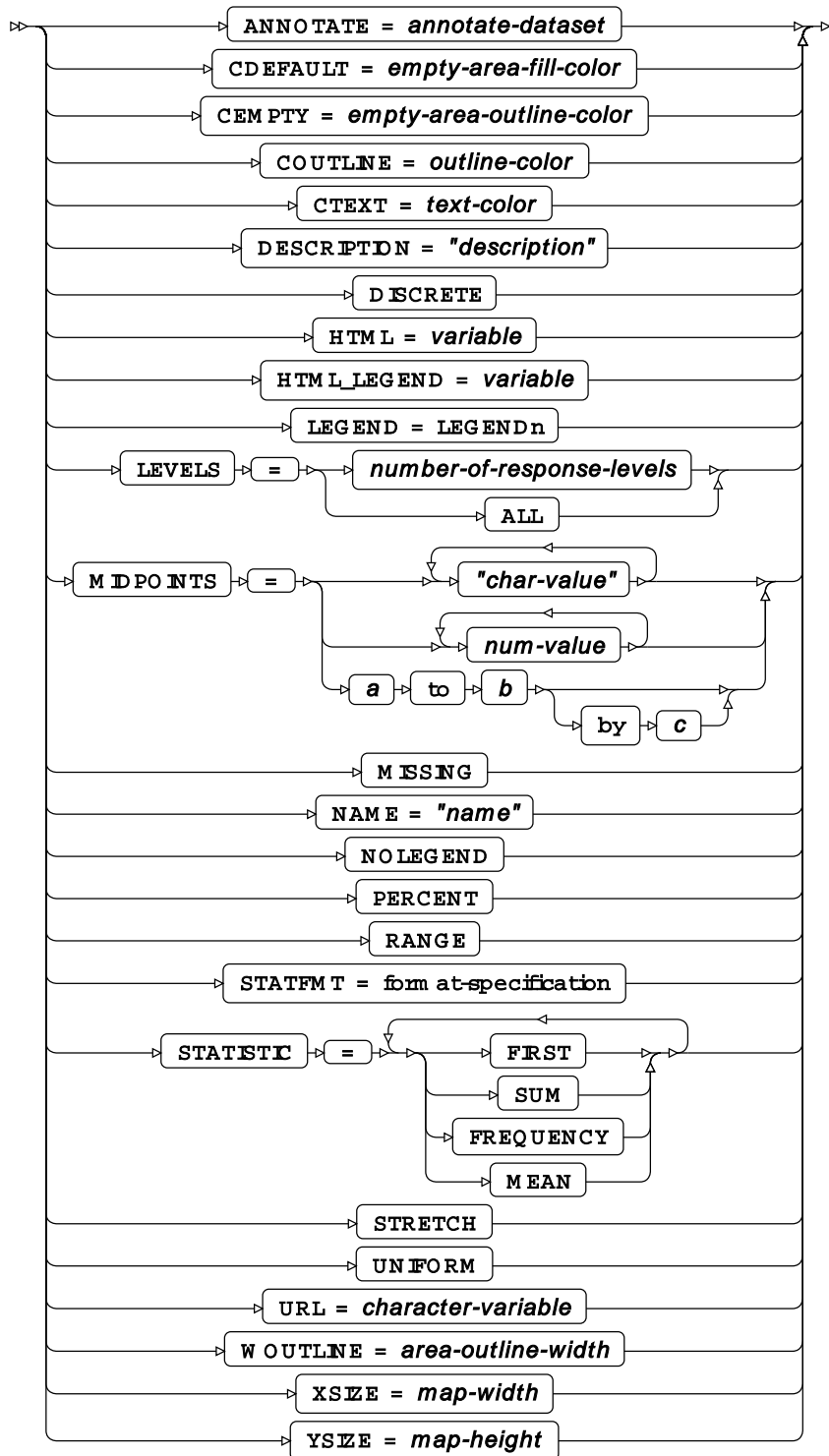


CHORO

Creates a region-coloured 2D map where the regions are coloured to the specified response variable.



options



FORMAT

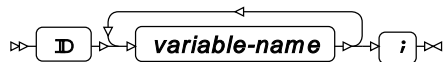
Adds formats to one or more variables in a dataset.



ⁱ See *Variable Lists* [↗](#) (page 32).

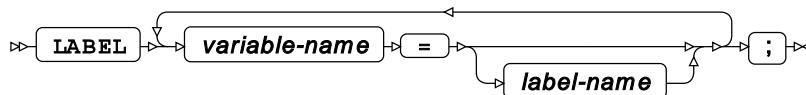
ID

Specifies the variables in a map dataset that define map areas.



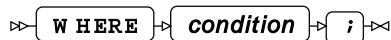
LABEL

Adds labels to one or more variables in a dataset.



WHERE

Restricts the observations to be processed.



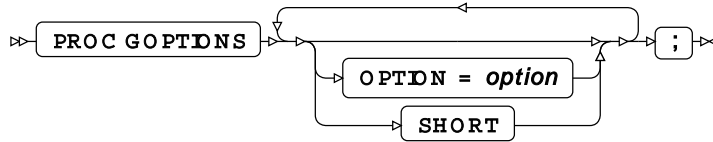
GOPTIONS procedure

Supported statements

- *PROC GOPTIONS* [↗](#) (page 2631)

PROC GOPTIONS

Provides the current status of graphic options and graphic-related global statements.



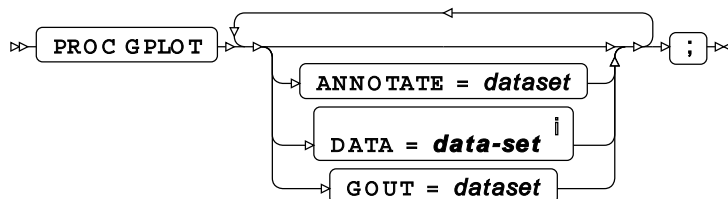
GPLOT procedure

Supported statements

- *PROC GPLOT* [↗](#) (page 2631)
- *BUBBLE* [↗](#) (page 2632)
- *BUBBLE2* [↗](#) (page 2634)
- *BY* [↗](#) (page 2636)
- *FORMAT* [↗](#) (page 2636)
- *LABEL* [↗](#) (page 2636)
- *PLOT* [↗](#) (page 2636)
- *PLOT2* [↗](#) (page 2638)
- *WHERE* [↗](#) (page 2639)

PROC GPLOT

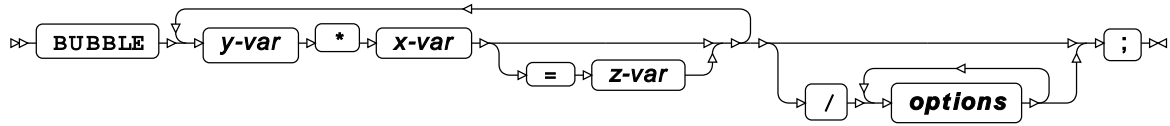
Plots pairs of variables against each other. Various interpolation methods can be used to connect the plot data.



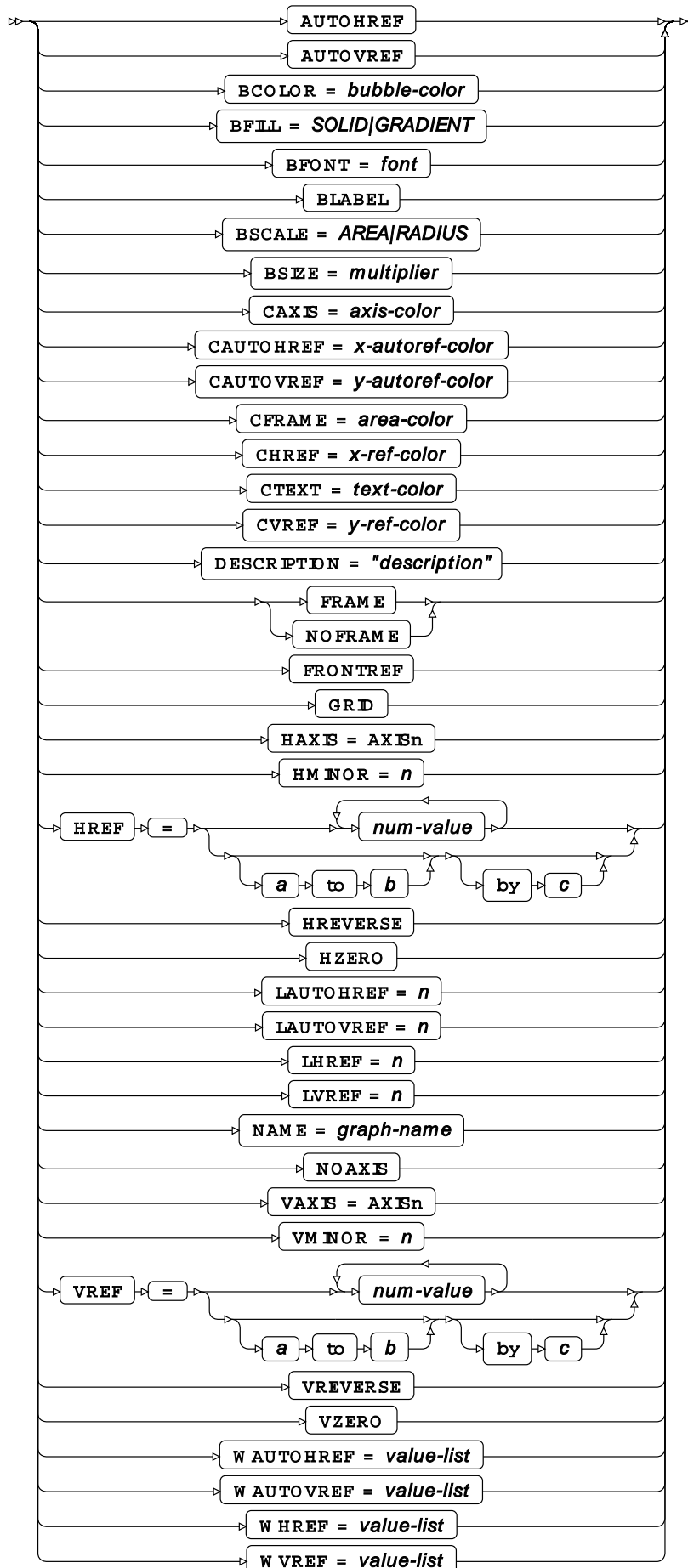
ⁱ See *Input dataset* [↗](#) (page 16).

BUBBLE

Creates one or more bubble plots where the variables are plotted on the horizontal and left-hand side vertical axis.

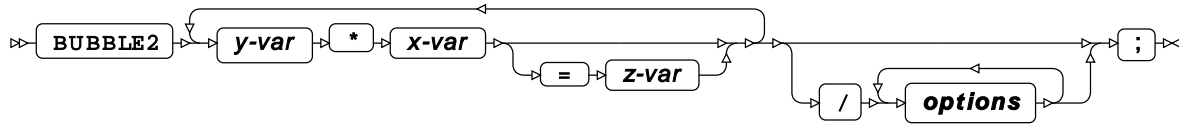


options

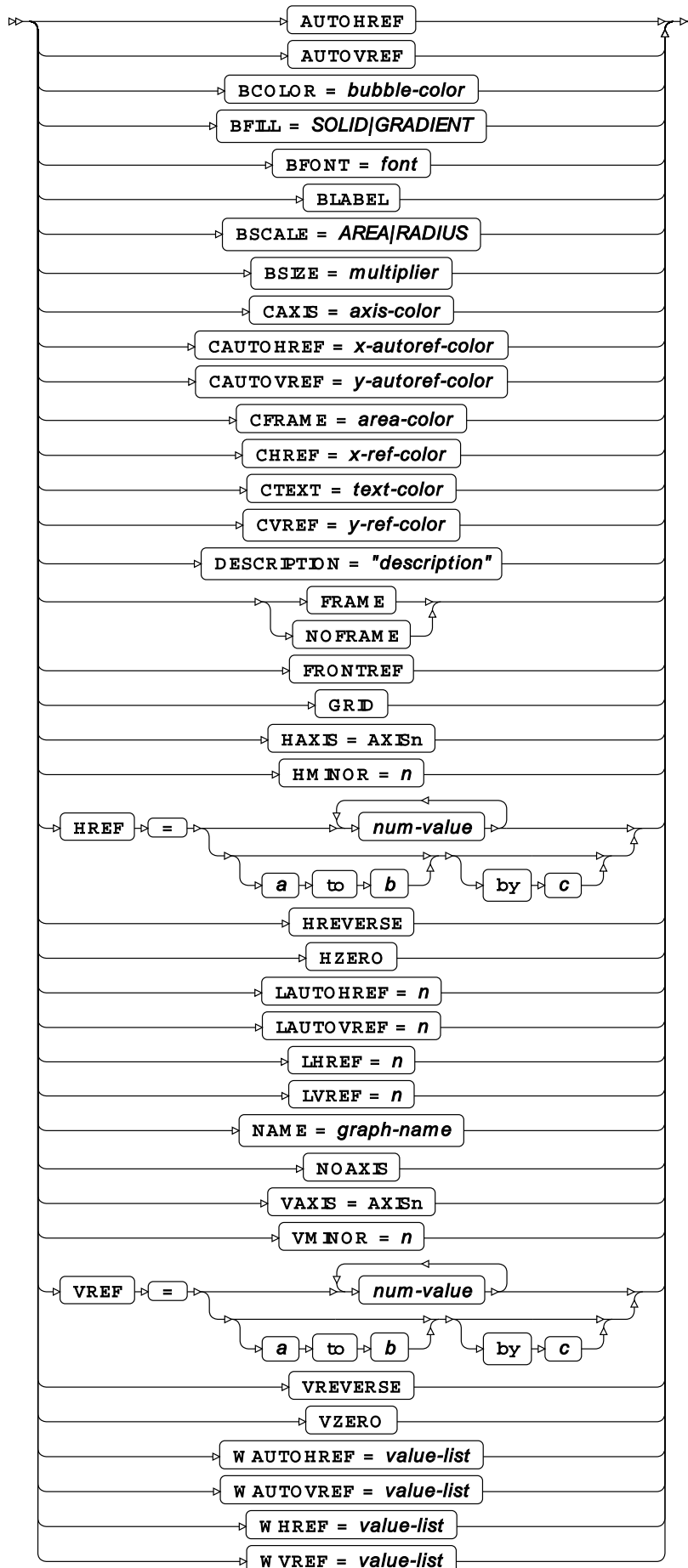


BUBBLE2

Creates one or more bubble plots where the variables are plotted on the horizontal and right-hand side vertical axis.

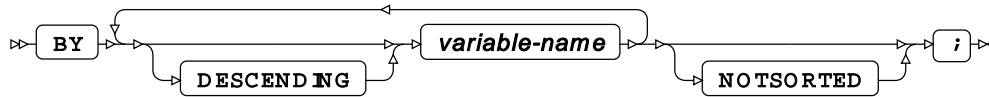


options



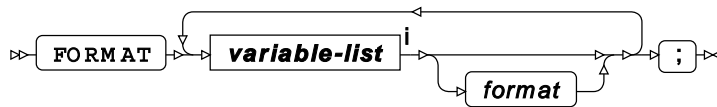
BY

Groups the observations in a dataset using one or more specified variables.



FORMAT

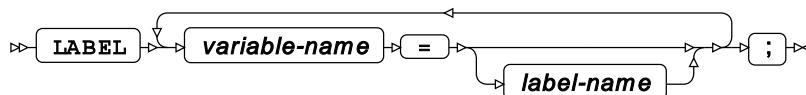
Adds formats to one or more variables in a dataset.



ⁱ See *Variable Lists* [↗](#) (page 32).

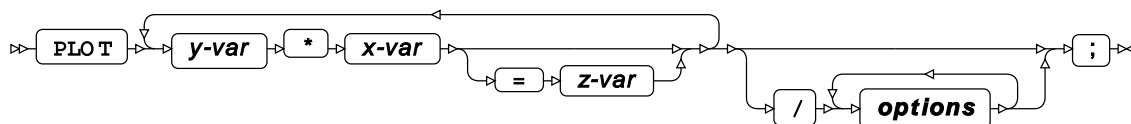
LABEL

Adds labels to one or more variables in a dataset.

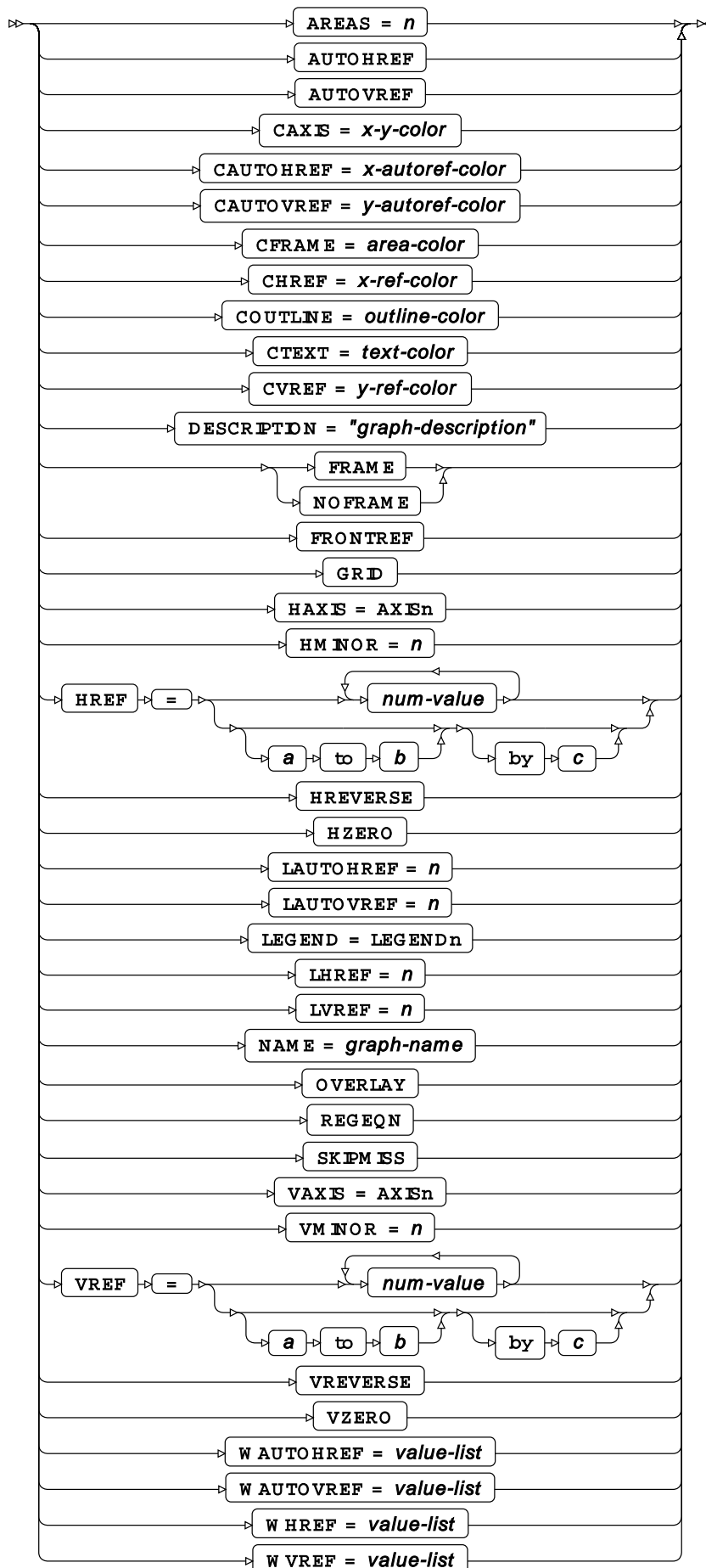


PLOT

Creates one or more plots. The variables are plotted on the horizontal and left-hand side vertical axis.

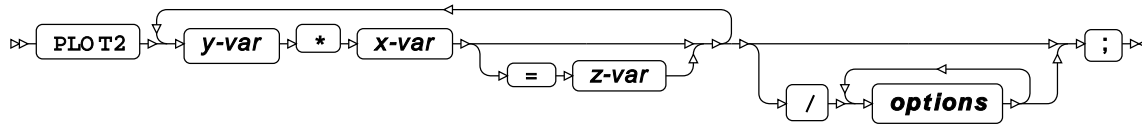


options

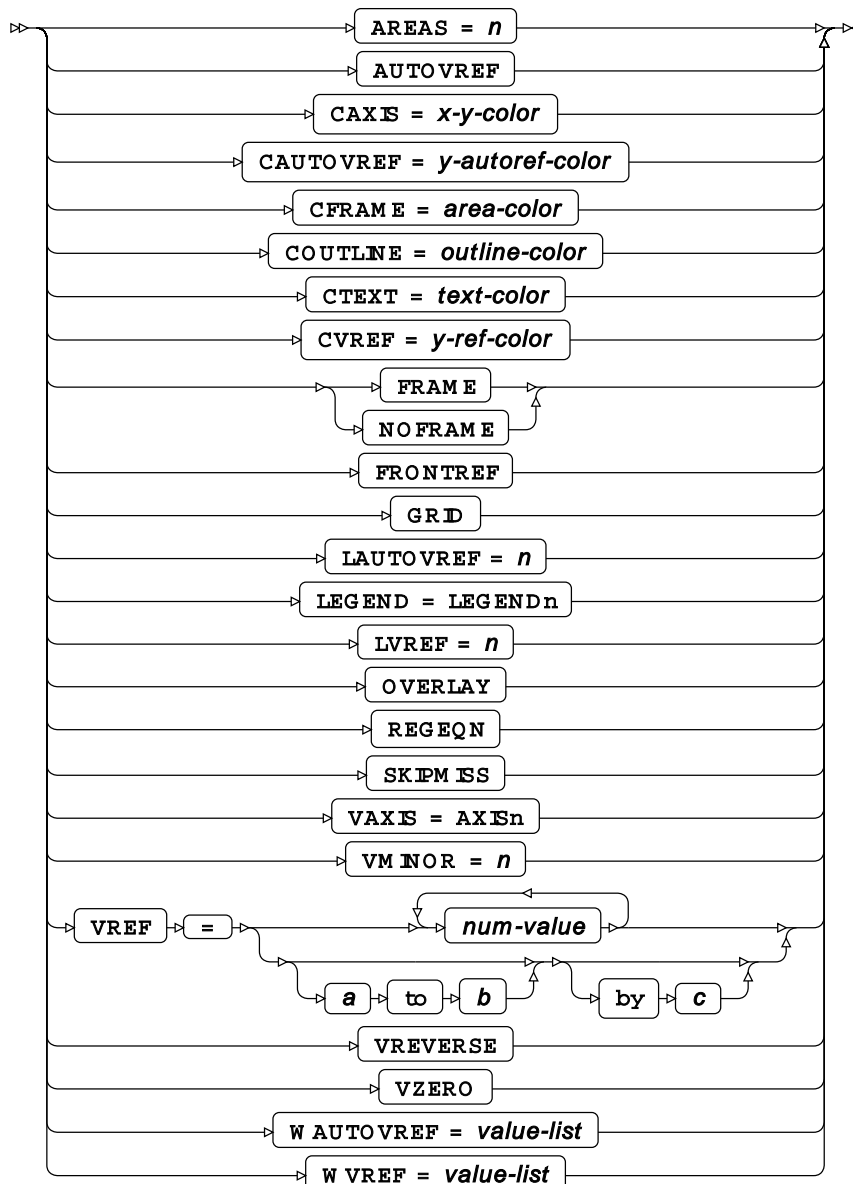


PLOT2

Creates one or more plots. The variables are plotted on the horizontal and right-hand side vertical axis.

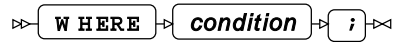


options



WHERE

Restricts the observations to be processed.



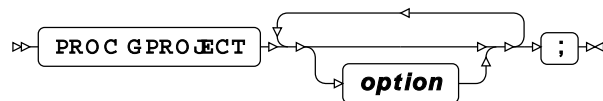
GPROJECT procedure

Supported statements

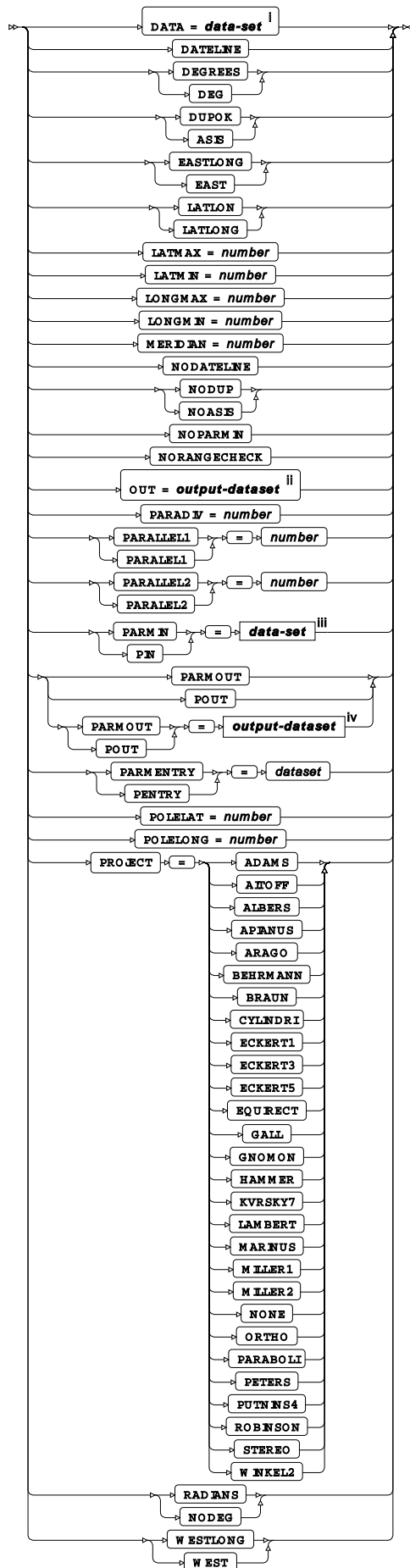
- *PROC GPROJECT* [↗](#) (page 2639)
- *ID* [↗](#) (page 2641)
- *WHERE* [↗](#) (page 2641)

PROC GPROJECT

Transforms map coordinates to a two-dimensional projected space.



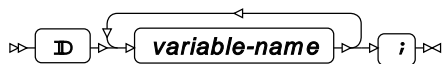
Option



- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).
- ^{iv} See *Output dataset* [↗](#) (page 16).

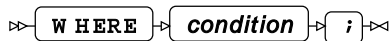
ID

Specifies the variables in a map dataset that define map areas.



WHERE

Restricts the observations to be processed.



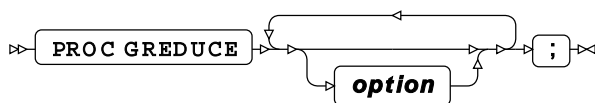
GREDUCE procedure

Supported statements

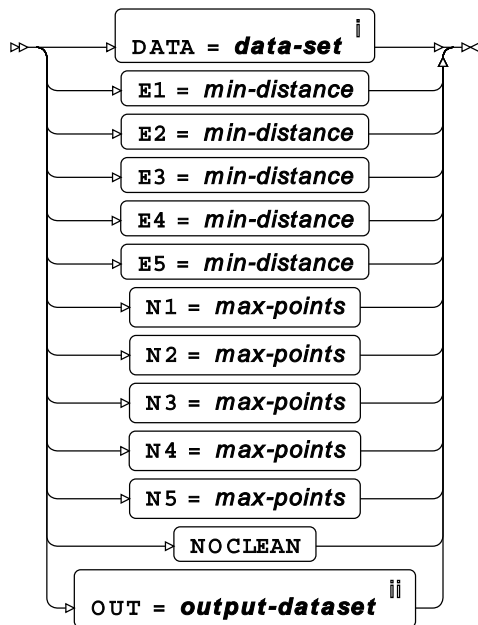
- *PROC GREDUCE* [↗](#) (page 2641)
- *BY* [↗](#) (page 2642)
- *ID* [↗](#) (page 2642)

PROC GREDUCE

Creates a low-resolution map from the specified map dataset using on a priority variable added to the output map dataset.



option

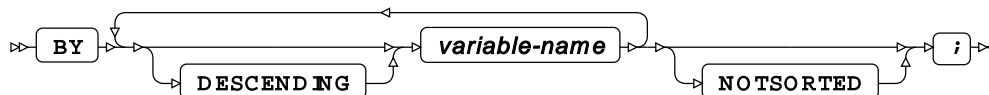


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

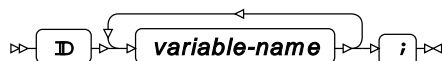
BY

Groups the observations in a dataset using one or more specified variables.



ID

Specifies the variables in a map dataset that define map areas.



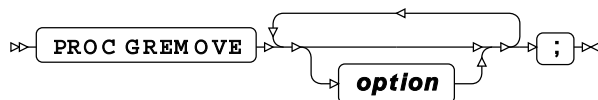
GREMOVE procedure

Supported statements

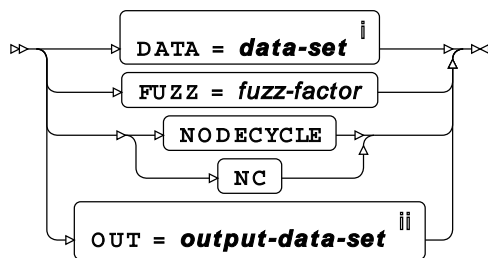
- *PROC GREMOVE* [↗](#) (page 2643)
- *BY* [↗](#) (page 2643)
- *ID* [↗](#) (page 2644)

PROC GREMOVE

Appends a density variable to the map dataset. The density variable can be used in the *GMAP* procedure to draw a lower-resolution version of the map.



option

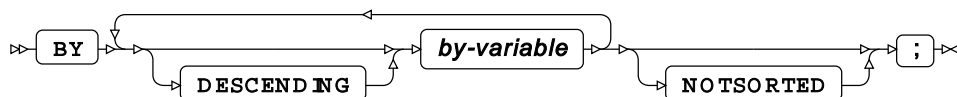


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

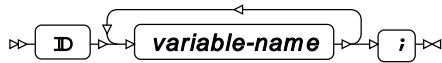
BY

Specifies the variables in a map dataset that define regions (a group of areas).



ID

Specifies the variables in a map dataset that define map areas.



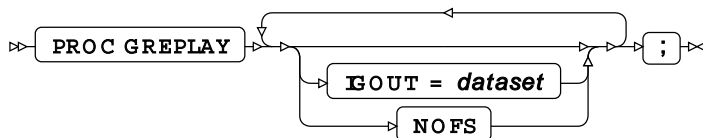
GREPLAY procedure

Supported statements

- *PROC GREPLAY* [↗](#) (page 2644)
- *DEVICE* [↗](#) (page 2644)
- *DELETE* [↗](#) (page 2645)
- *IGOUT* [↗](#) (page 2645)
- *LIST* [↗](#) (page 2645)
- *QUIT* [↗](#) (page 2645)
- *REPLAY* [↗](#) (page 2646)

PROC GREPLAY

Replays graphics output entries that are stored in a WPS catalog.



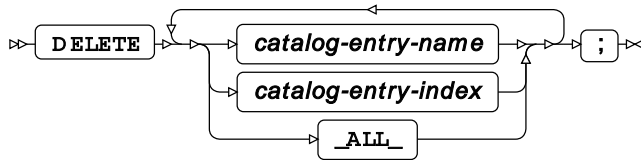
DEVICE

Specifies the device driver to use when using the GREPLAY procedure.



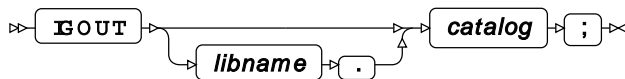
DELETE

Removes graphic entries from the catalog.



IGOUT

Specifies the input catalog for use.



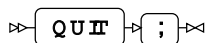
LIST

Lists the graphic entries that exist in the graphics catalog.



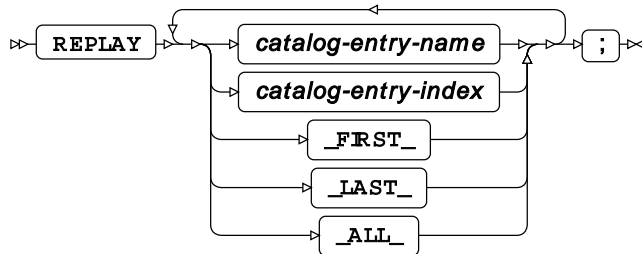
QUIT

Quits the procedure.



REPLAY

Selects and replays the graphics entries in the input catalog.



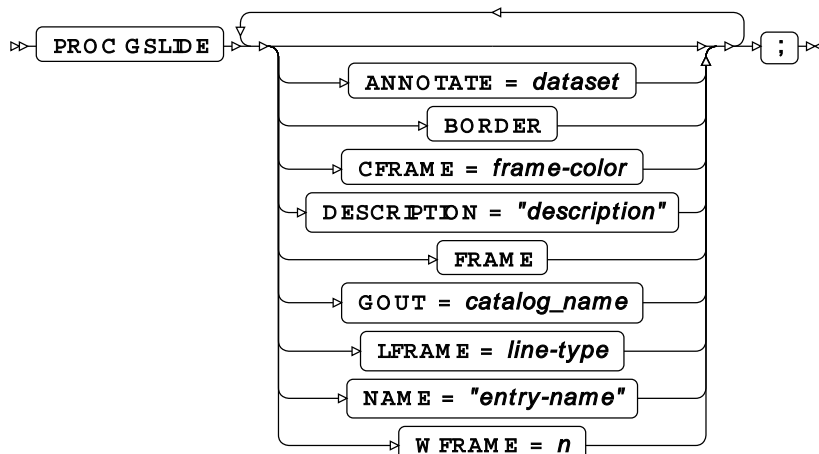
GSLIDE procedure

Supported statements

- **PROC GSLIDE** [↗](#) (page 2646)

PROC GSLIDE

Adds header and footer information to graphical output from an ANNOTATE dataset.



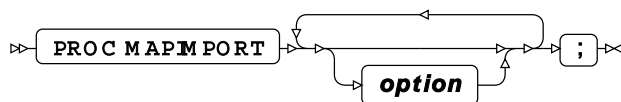
MAPIMPORT procedure

Supported statements

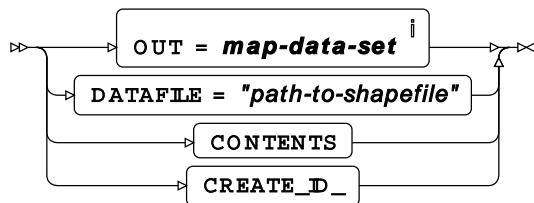
- *PROC MAPIMPORT* [↗](#) (page 2647)
- *ID* [↗](#) (page 2647)
- *EXCLUDE* [↗](#) (page 2648)
- *RENAME* [↗](#) (page 2648)
- *SELECT* [↗](#) (page 2648)

PROC MAPIMPORT

Imports ESRI *shapefile* data into a WPS map dataset.



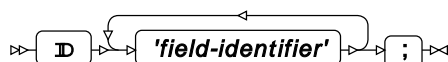
option



ⁱ See *Output dataset* [↗](#) (page 16).

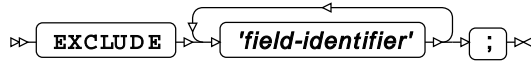
ID

Specifies the variables in a map dataset that define map areas, and reorders the map areas based on the specified variables.



EXCLUDE

Specifies one or more field identifier that are excluded from the output map dataset.



RENAME

Renames a specified field identifier in the output map dataset.



SELECT

Specifies one or more field identifier that are included in the output map dataset. An excluded field identifier cannot be specified in the **SELECT** statement.



SGPANEL procedure

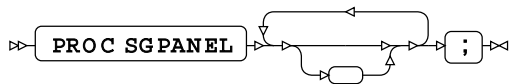
Supported statements

- *PROC SGPANEL* [↗](#) (page 2649)
- *BAND* [↗](#) (page 2650)
- *BUBBLE* [↗](#) (page 2652)
- *BY* [↗](#) (page 2654)
- *COLAXIS* [↗](#) (page 2654)
- *DENSITY* [↗](#) (page 2656)
- *FORMAT* [↗](#) (page 2658)
- *HBAR* [↗](#) (page 2658)
- *HBARPARM* [↗](#) (page 2660)
- *HBOX* [↗](#) (page 2662)
- *HIGHLOW* [↗](#) (page 2665)

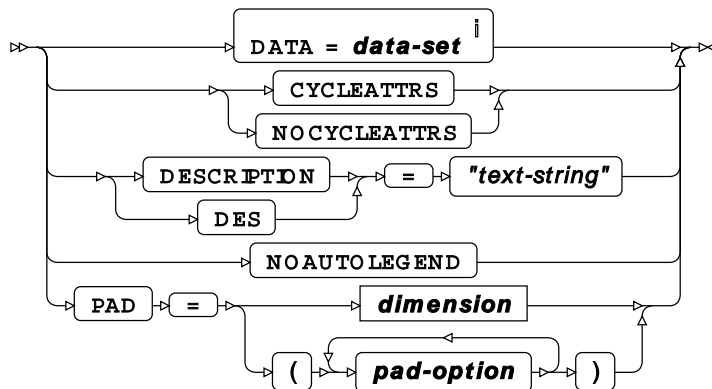
- *HISTOGRAM* [↗](#) (page 2667)
- *HLINE* [↗](#) (page 2668)
- *KEYLEGEND* [↗](#) (page 2670)
- *LABEL* [↗](#) (page 2672)
- *LINEPARM* [↗](#) (page 2672)
- *LOESS* [↗](#) (page 2673)
- *NEEDLE* [↗](#) (page 2676)
- *PANELBY* [↗](#) (page 2678)
- *PBSPLINE* [↗](#) (page 2679)
- *REFLINE* [↗](#) (page 2682)
- *REG* [↗](#) (page 2684)
- *ROWAXIS* [↗](#) (page 2687)
- *SCATTER* [↗](#) (page 2689)
- *SERIES* [↗](#) (page 2692)
- *STEP* [↗](#) (page 2695)
- *VBAR* [↗](#) (page 2698)
- *VBARPARM* [↗](#) (page 2700)
- *VBOX* [↗](#) (page 2702)
- *VECTOR* [↗](#) (page 2705)
- *VLINE* [↗](#) (page 2706)
- *WHERE* [↗](#) (page 2708)

PROC SG PANEL

Outputs multiple plots where each plot is determined by the class of data in the input dataset.

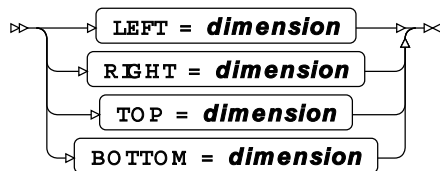


option

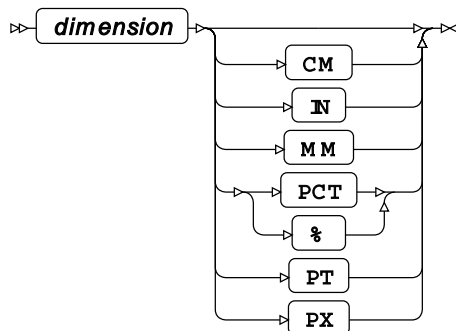


ⁱ See *Input dataset* [↗](#) (page 16).

pad-option

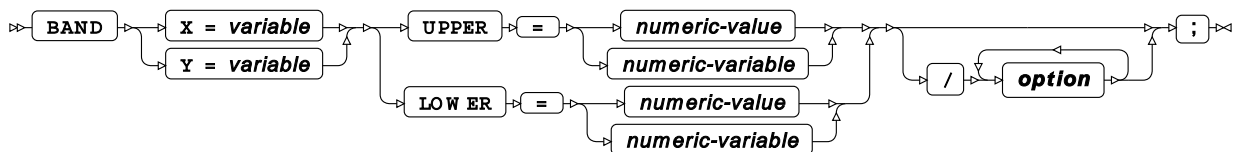


dimension

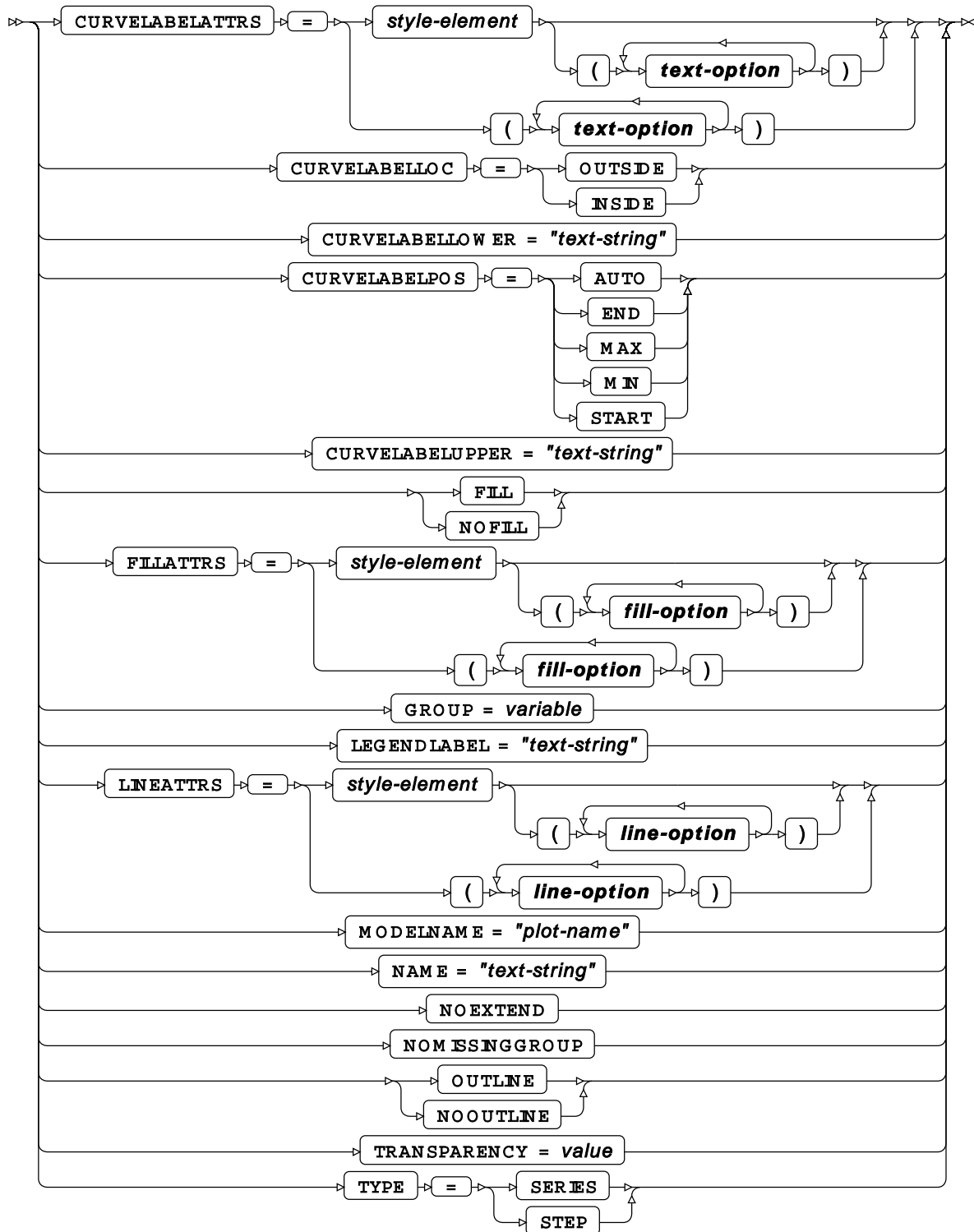


BAND

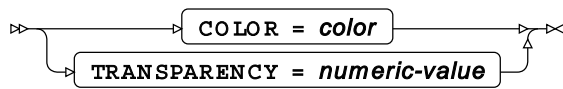
Draws band plots.



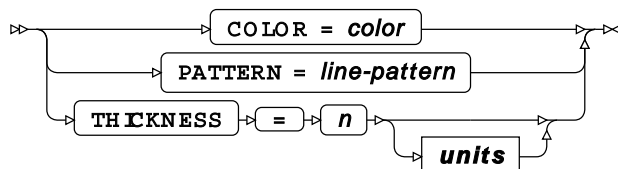
option



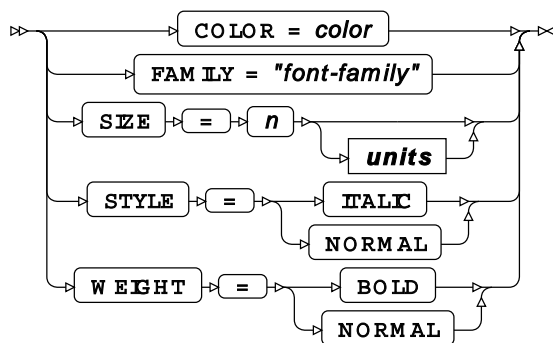
fill-option



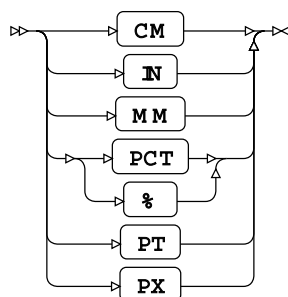
line-option



text-option

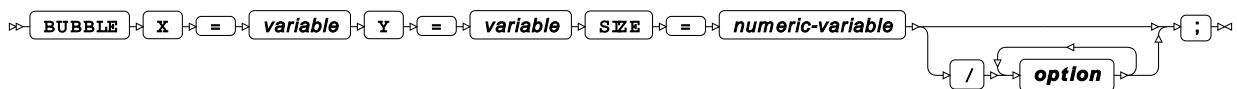


units

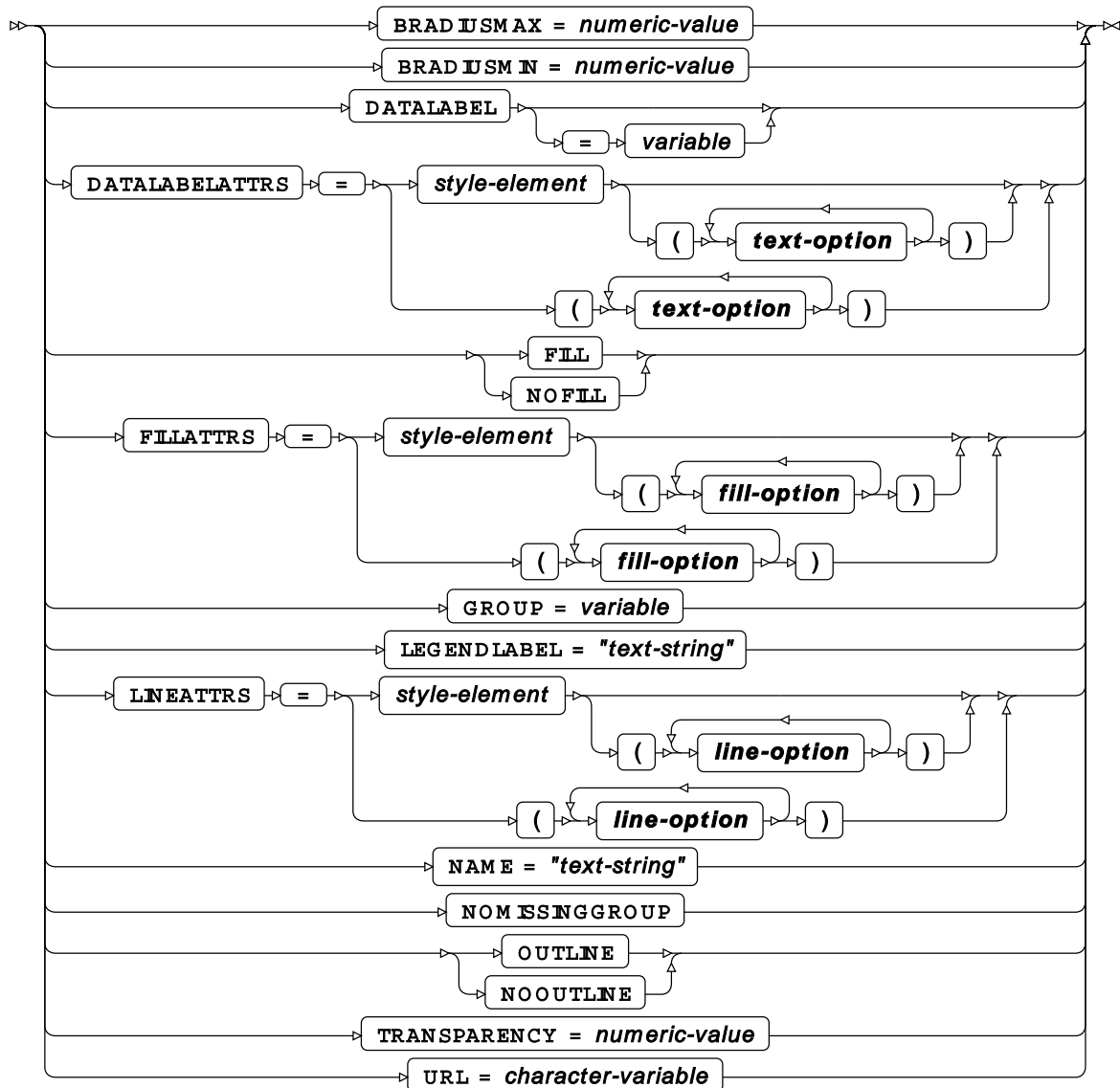


BUBBLE

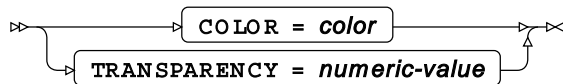
Draws bubble plots.



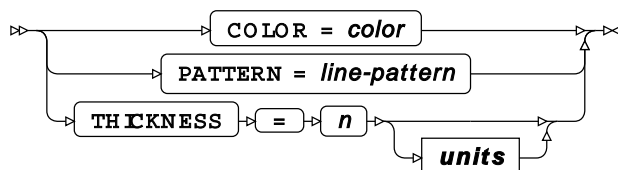
option



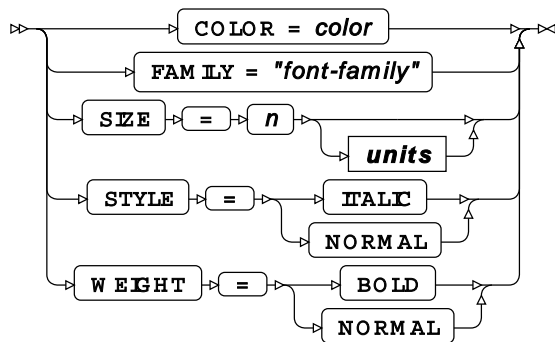
fill-option



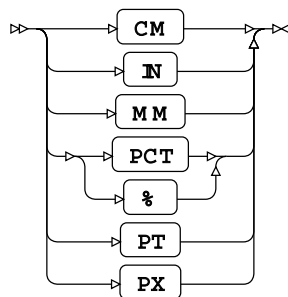
line-option



text-option

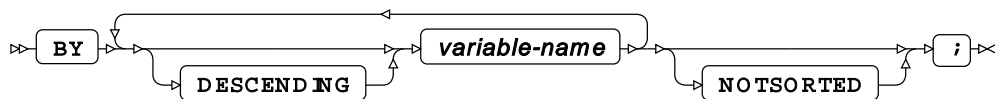


units



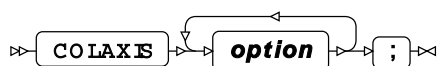
BY

Groups the observations in a dataset using one or more specified variables.

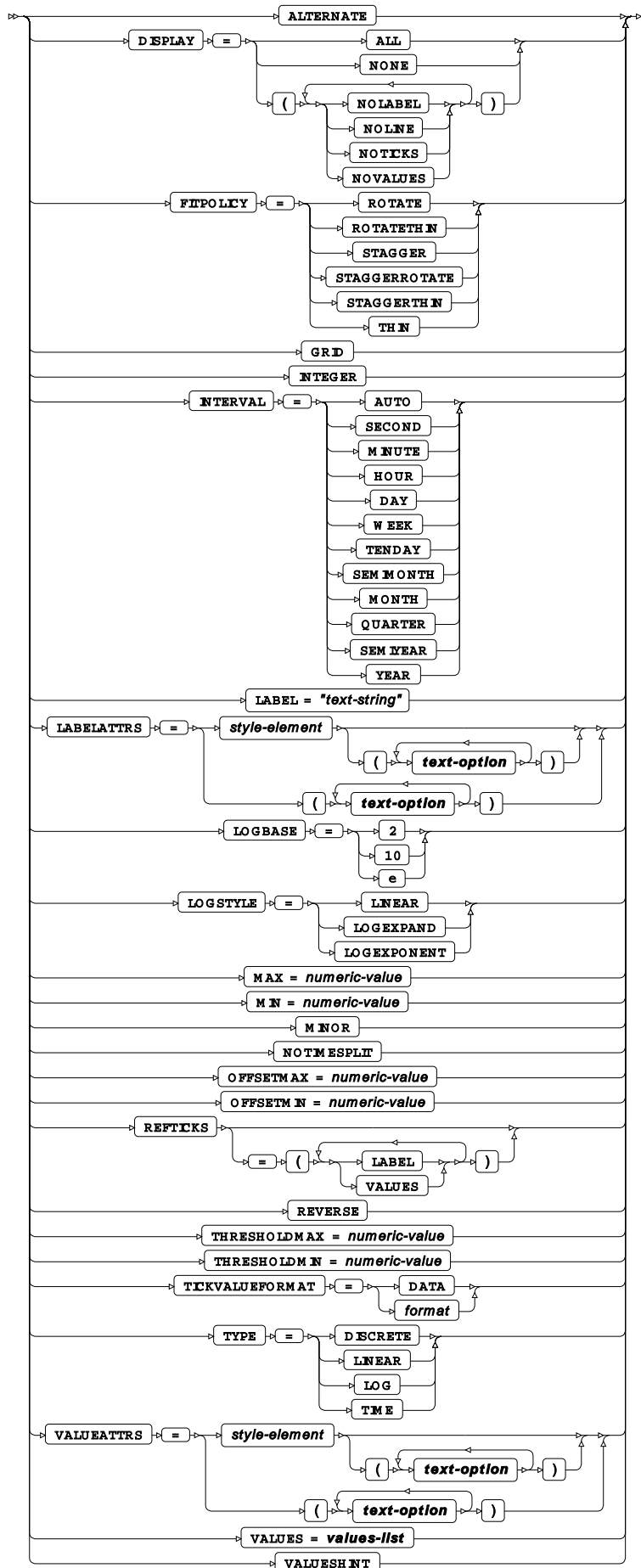


COLAXIS

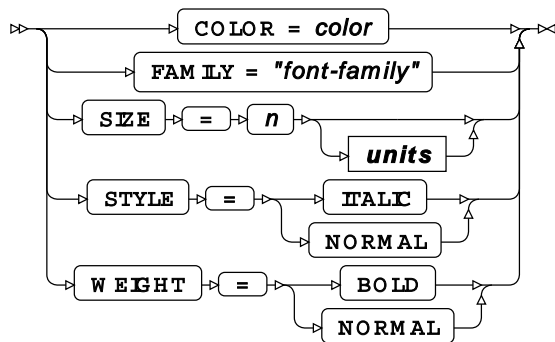
Specifies options for the column axes used in each graph.



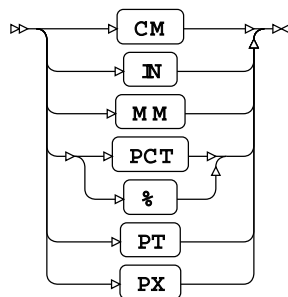
option



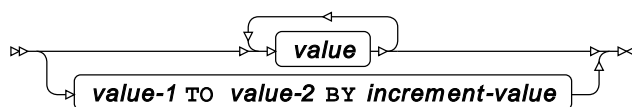
text-option



units

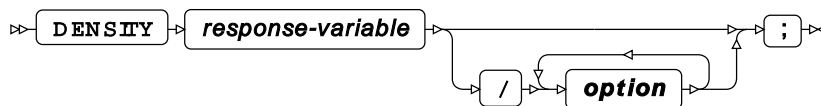


values-list

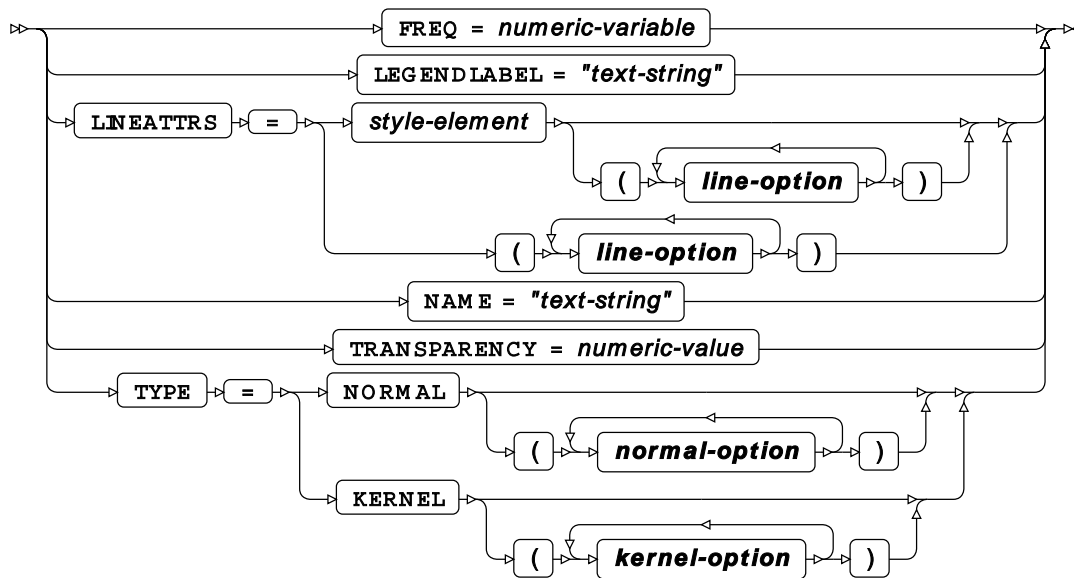


DENSITY

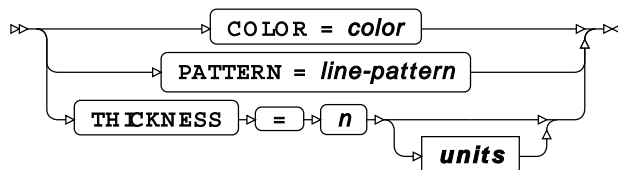
Draws density curves.



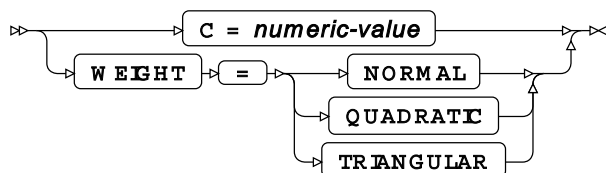
option



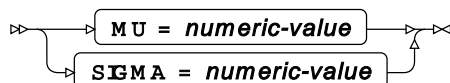
line-option



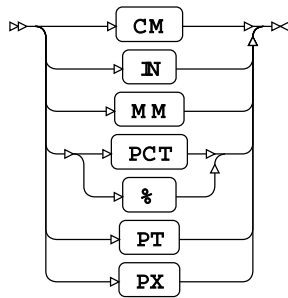
kernel-option



normal-option



units



FORMAT

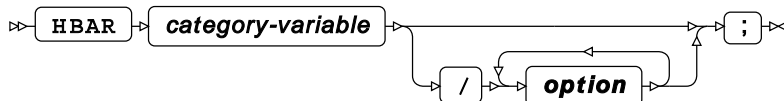
Adds formats to one or more variables in a dataset.



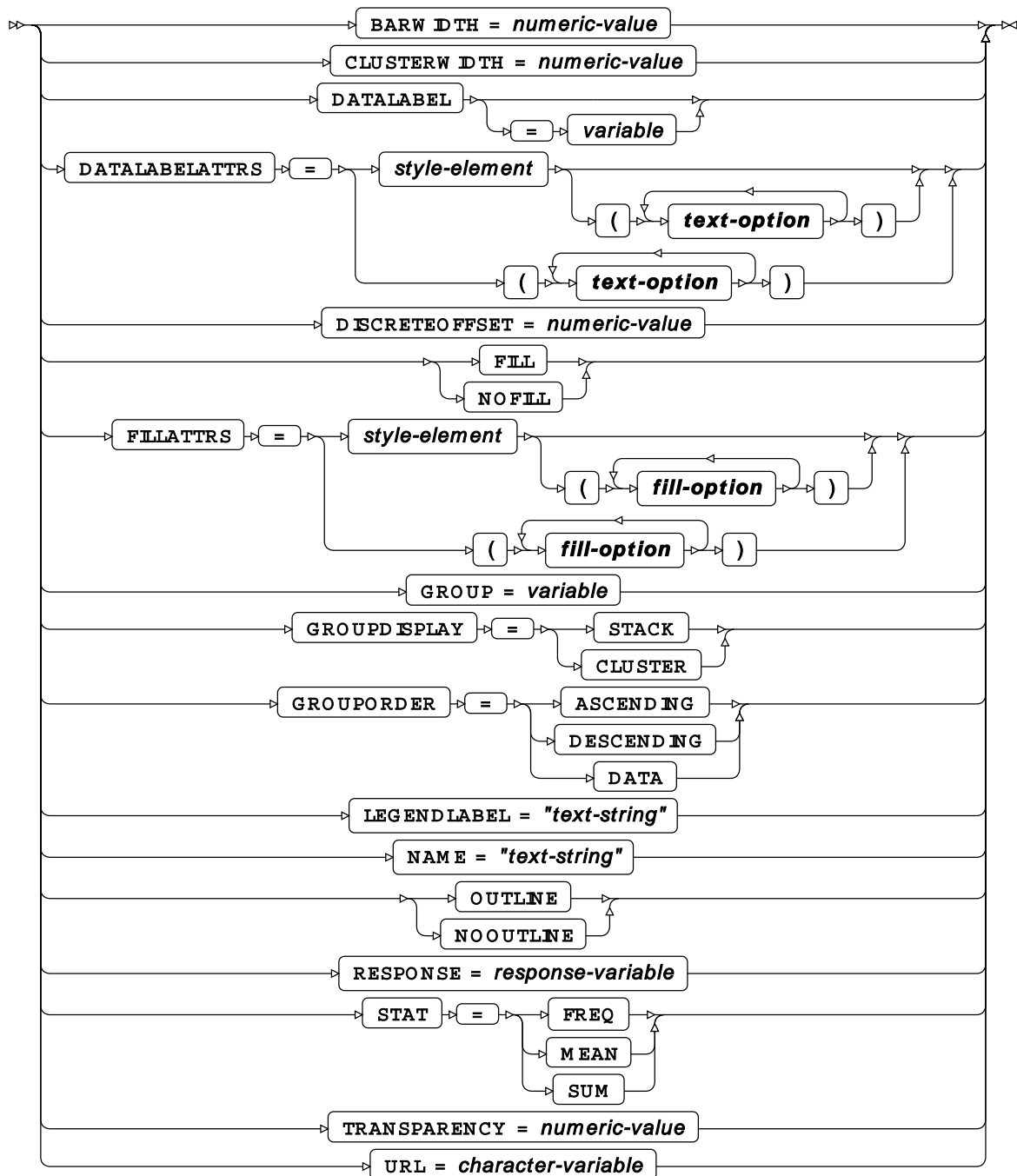
ⁱ See [Variable Lists](#) (page 32).

HBAR

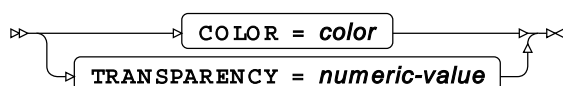
Draws horizontal bar charts using unsummarised data.



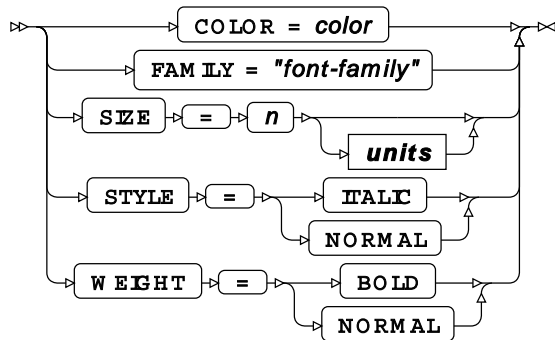
option



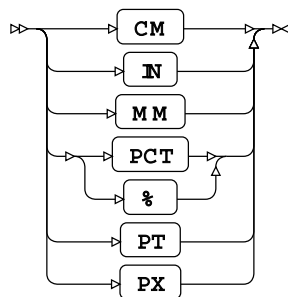
fill-option



text-option

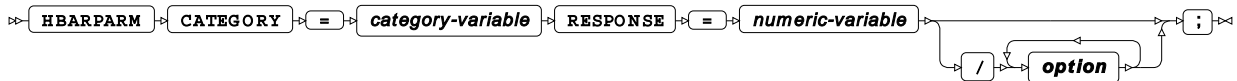


units

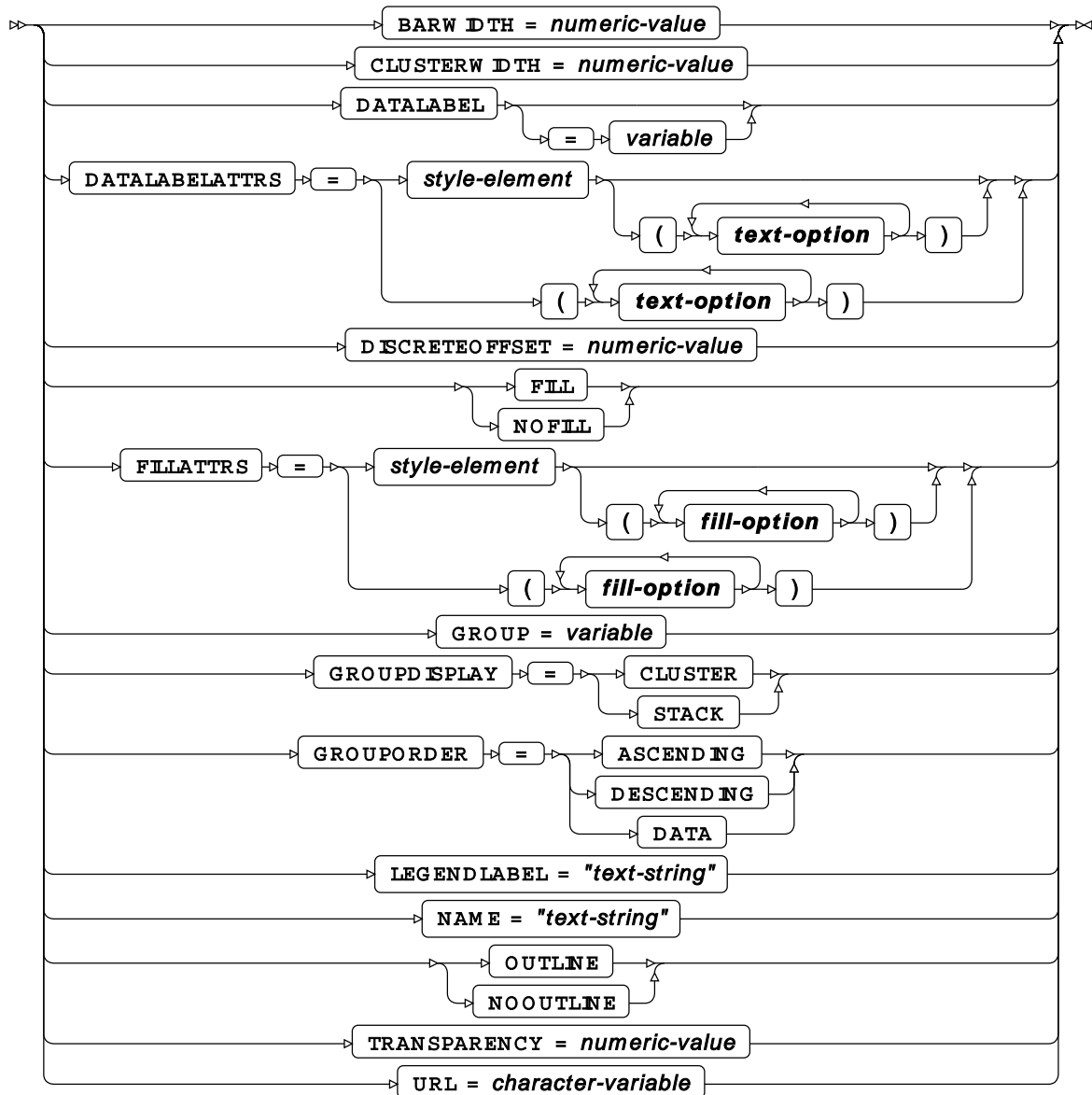


HBARPARM

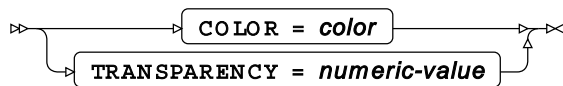
Draws horizontal bar charts using summarised data.



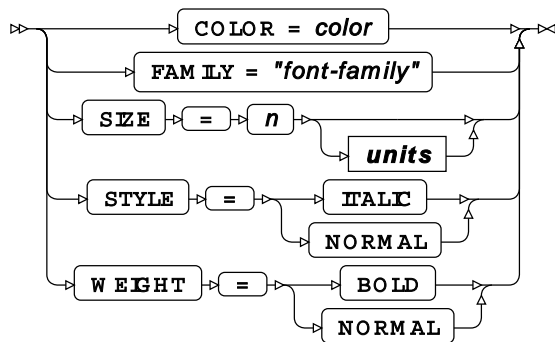
option



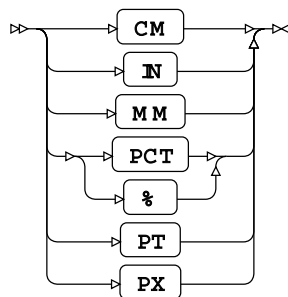
fill-option



text-option

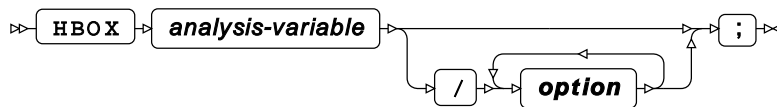


units

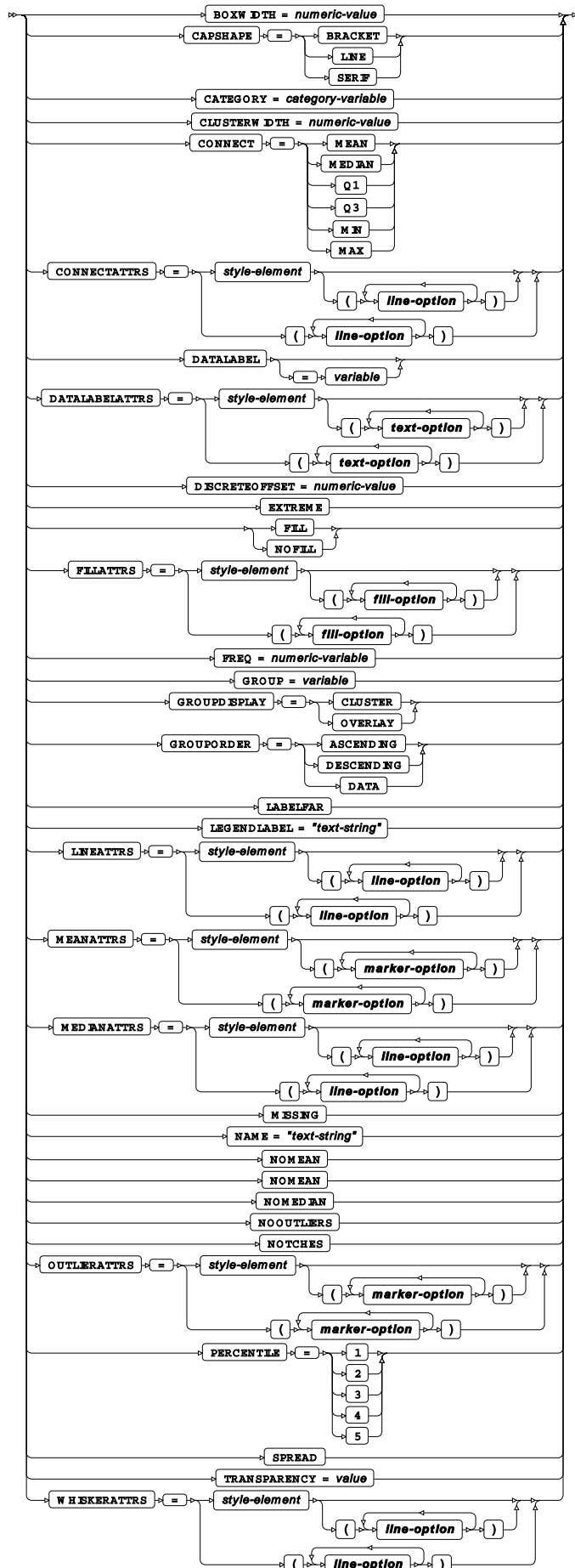


HBOX

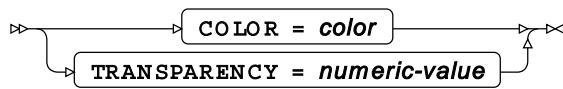
Draws horizontal box plots.



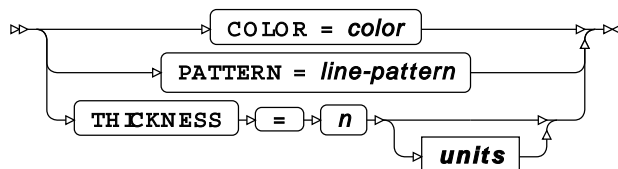
option



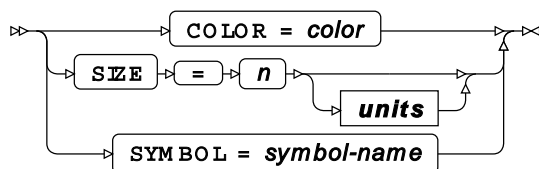
fill-option



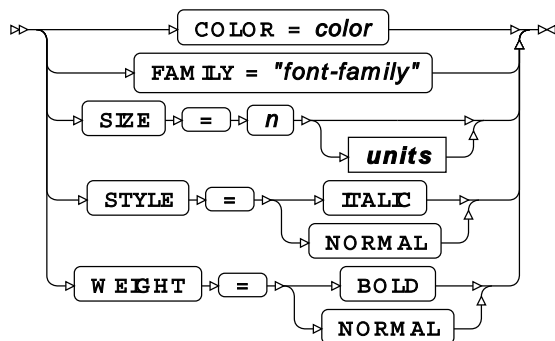
line-option



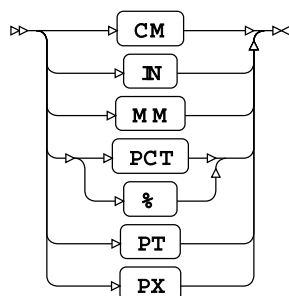
marker-option



text-option

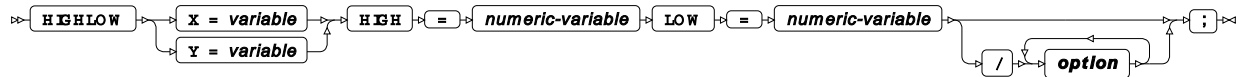


units

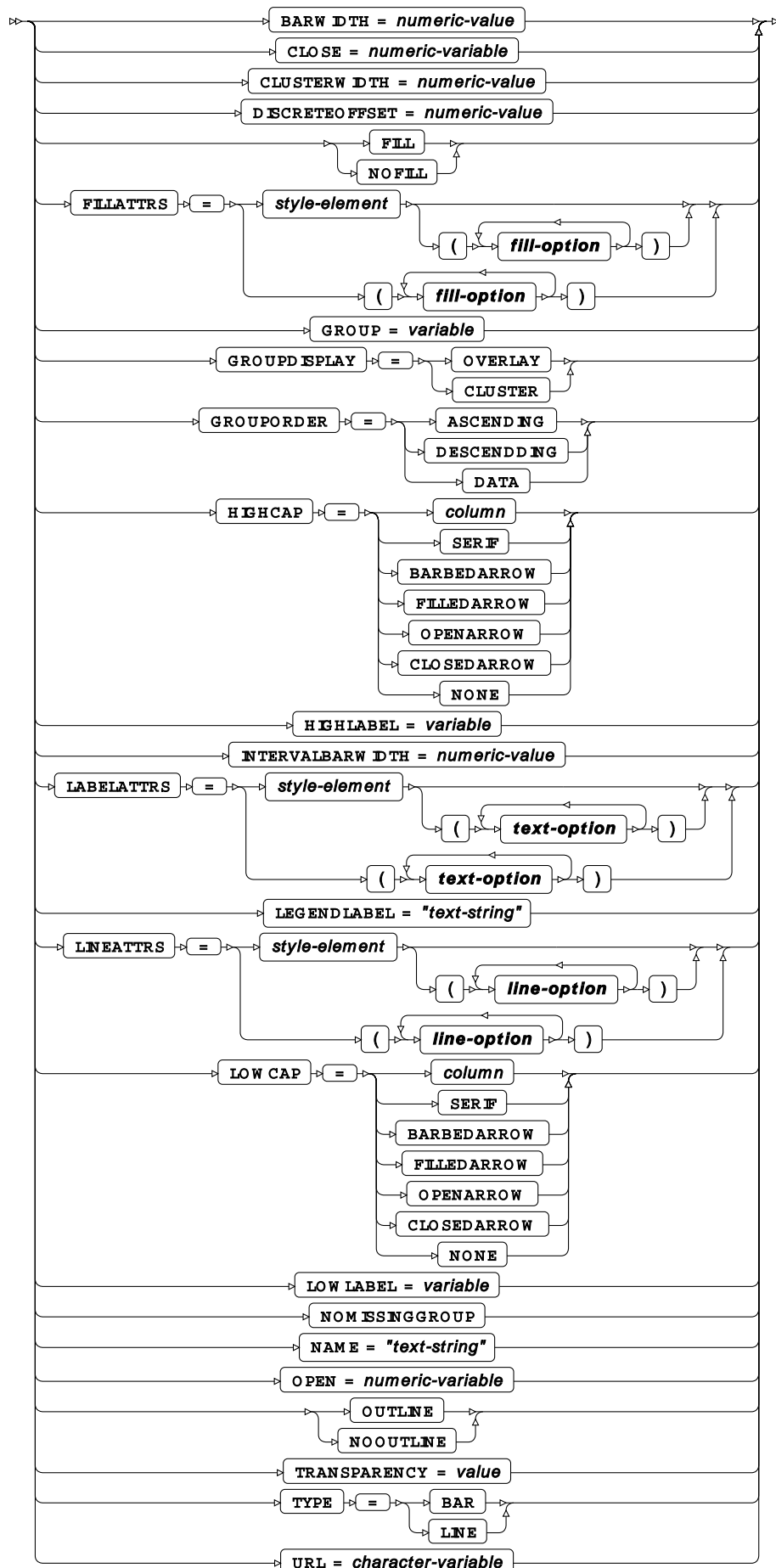


HIGHLOW

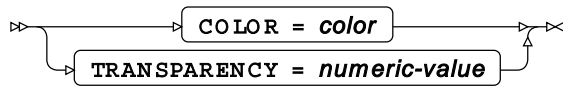
Draws high-low plots for categorised data.



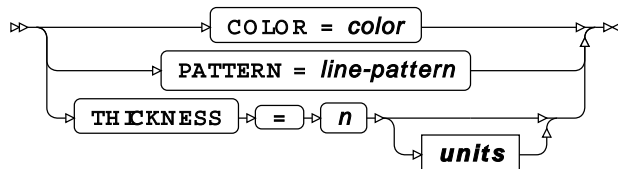
option



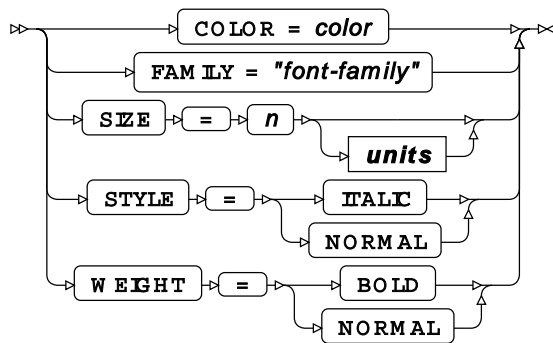
fill-option



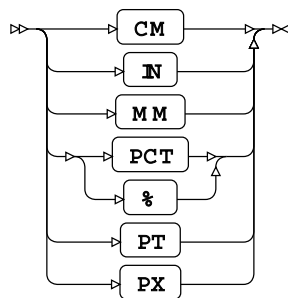
line-option



text-option

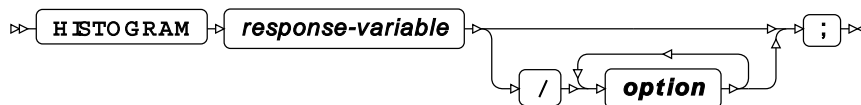


units

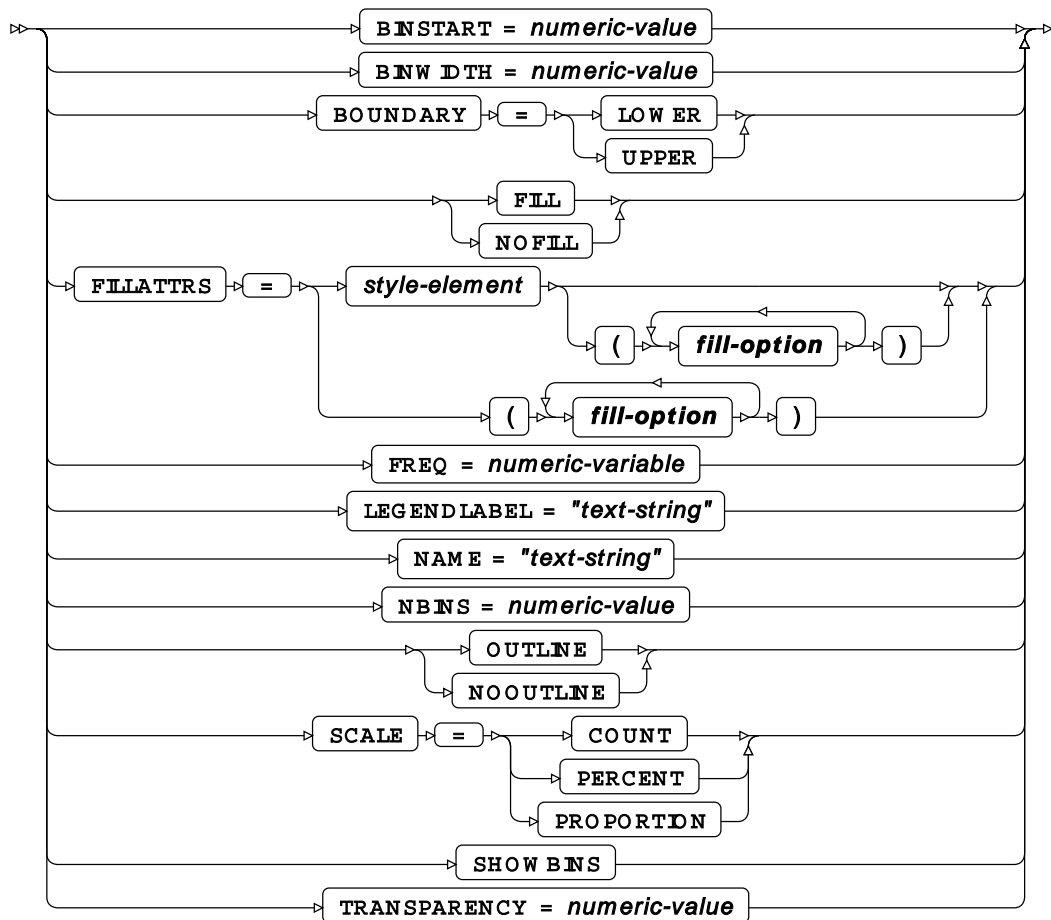


HISTOGRAM

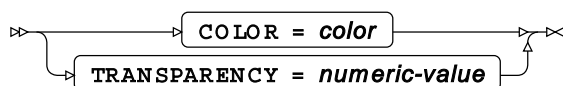
Draws histograms.



option

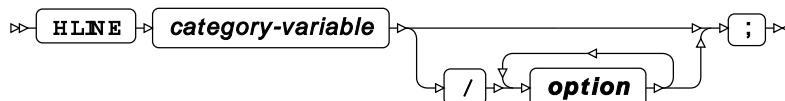


fill-option

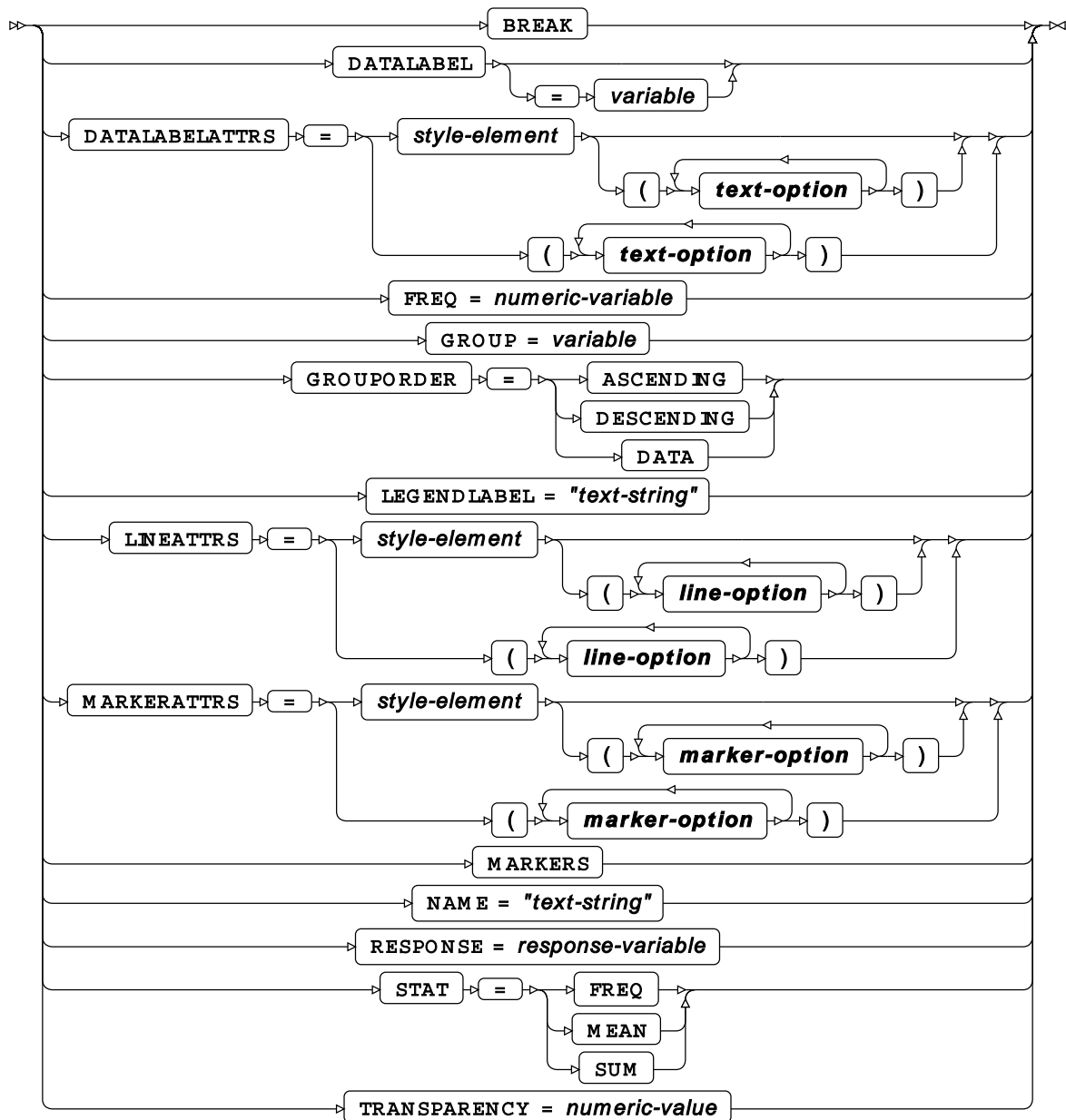


HLINE

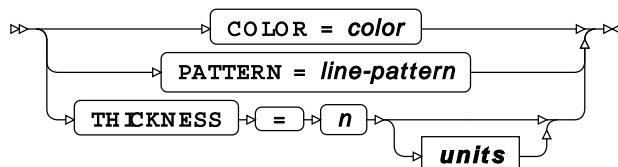
Draws horizontal line plots using unsummarised data.



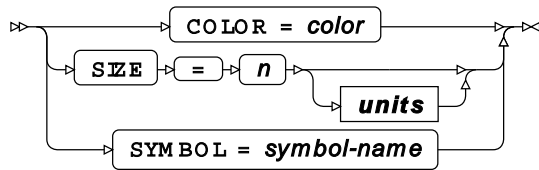
option



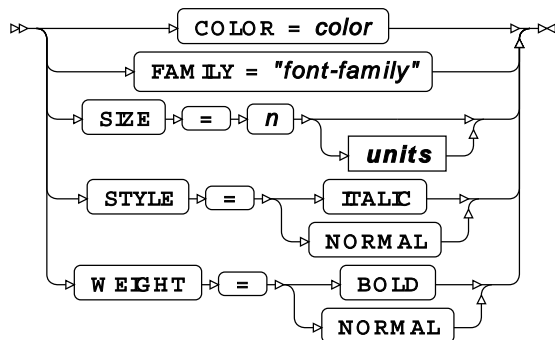
line-option



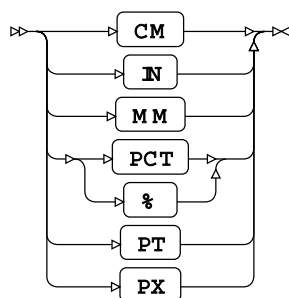
marker-option



text-option

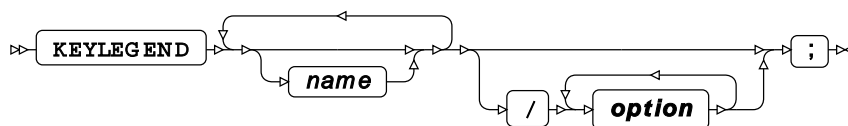


units

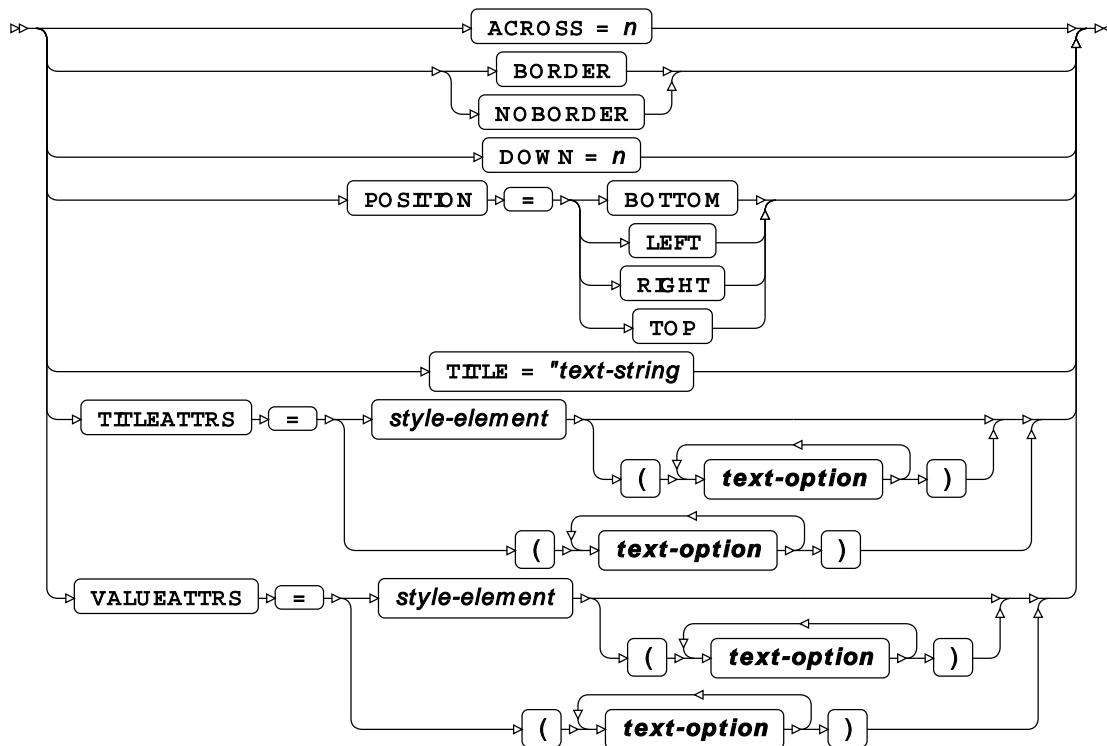


KEYLEGEND

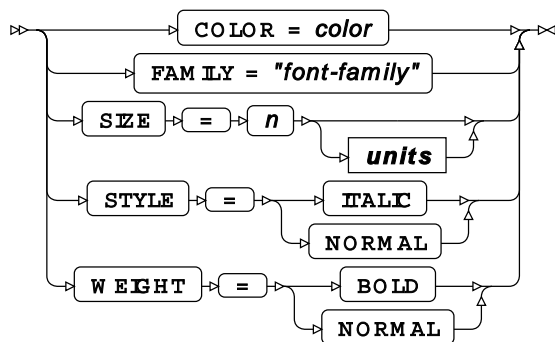
Specifies options for legends added to the plots.



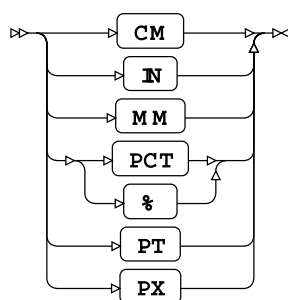
option



text-option

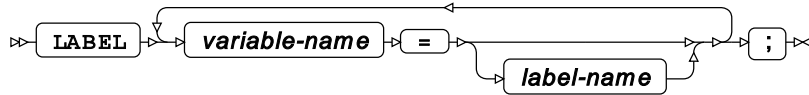


units



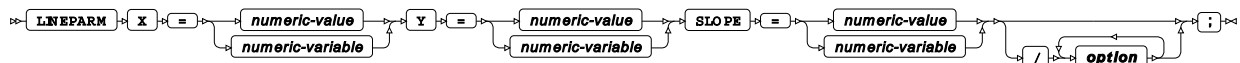
LABEL

Adds labels to one or more variables in a dataset.

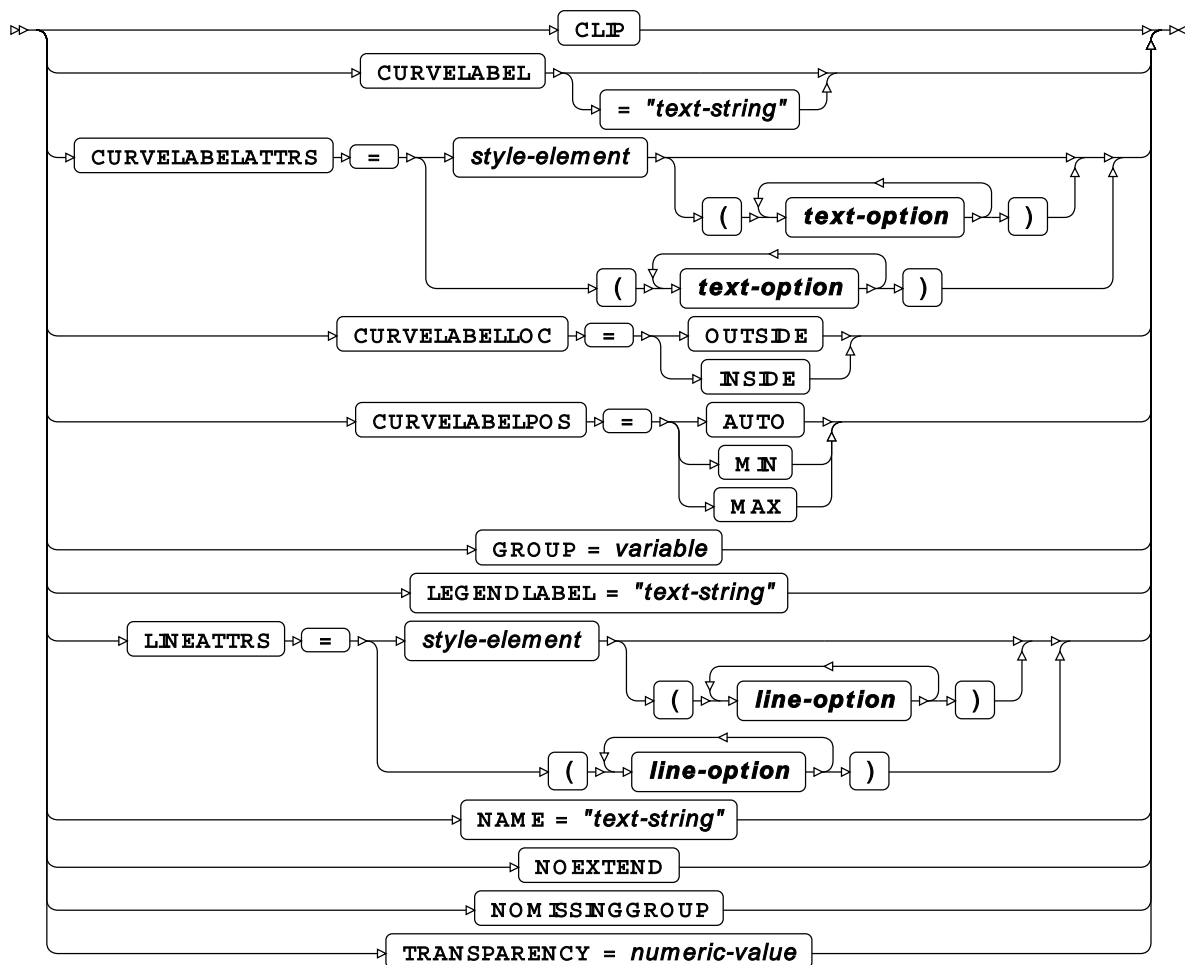


LINEPARM

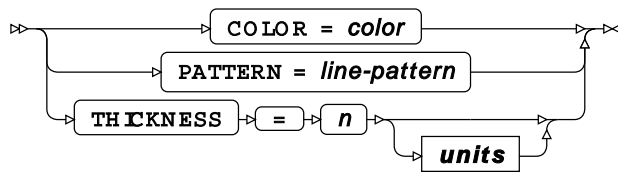
Draws one or more straight lines each defined by a point and gradient.



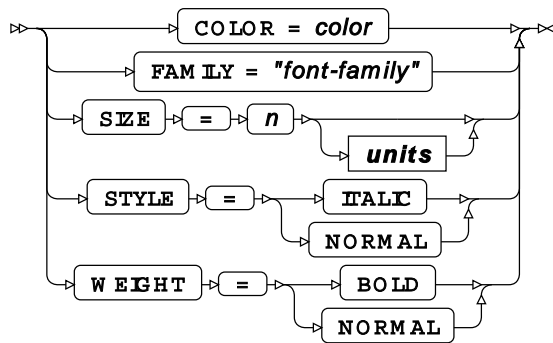
option



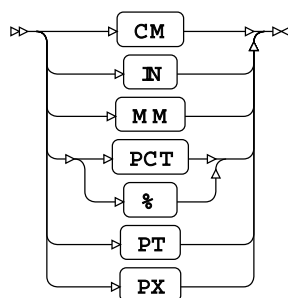
line-option



text-option

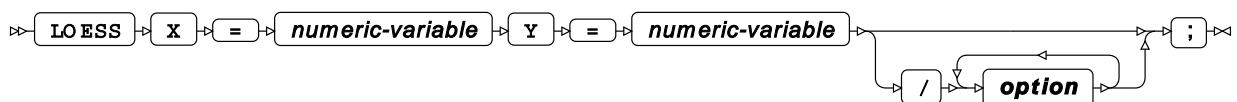


units

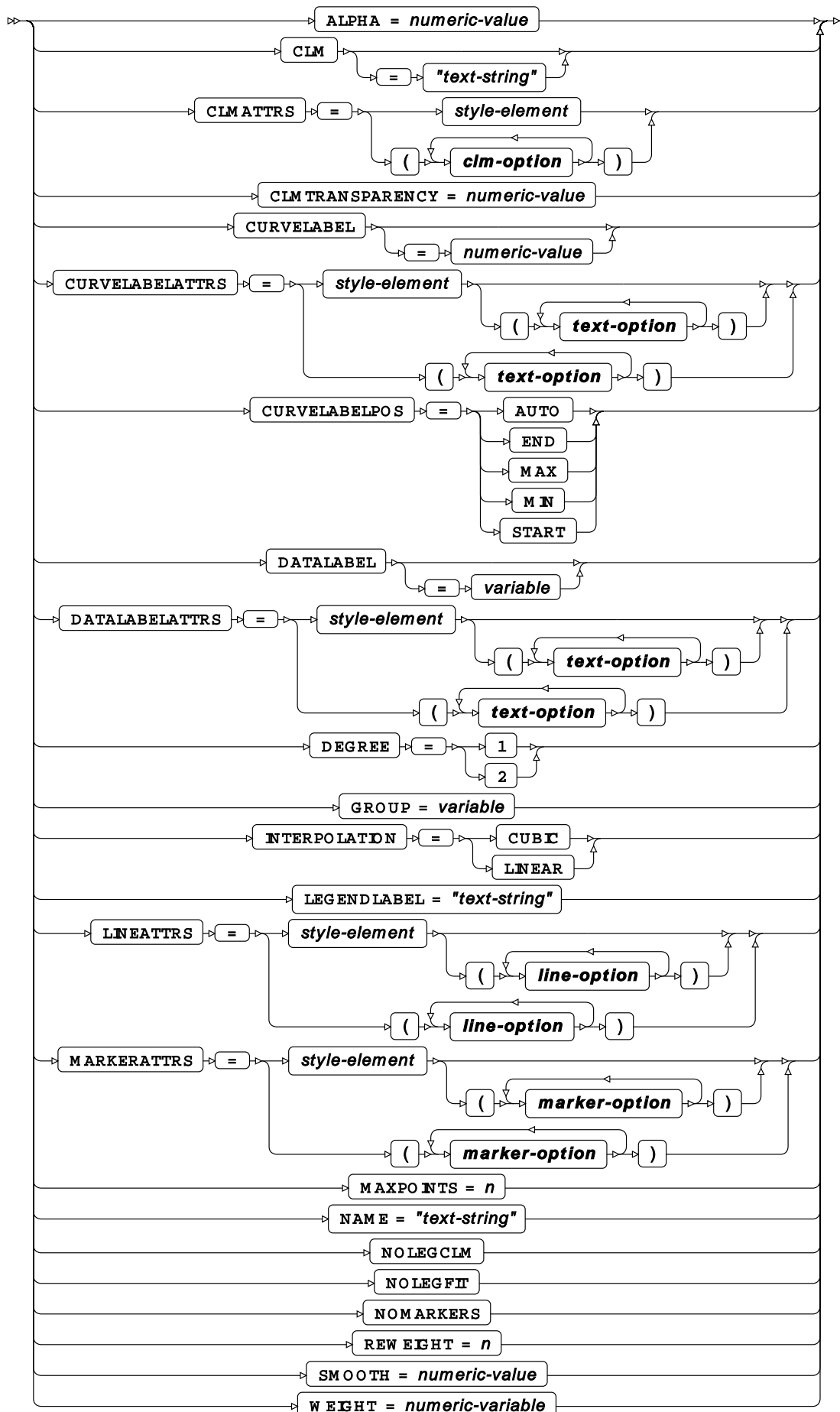


LOESS

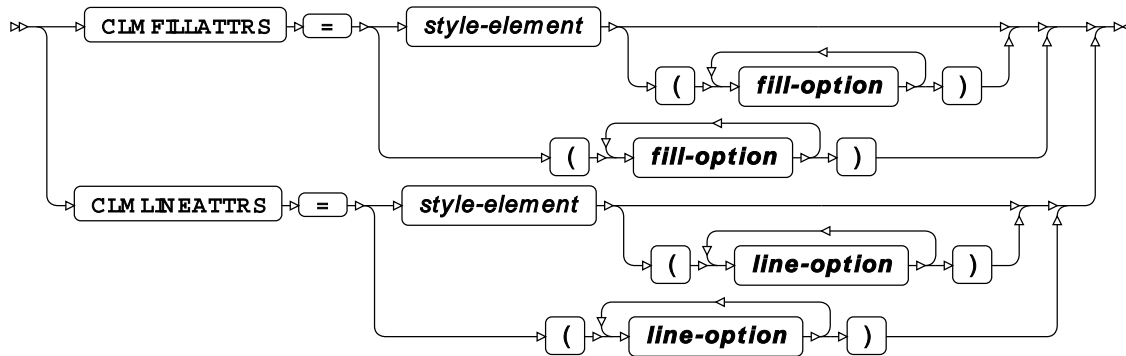
Draws fitted loess curves.



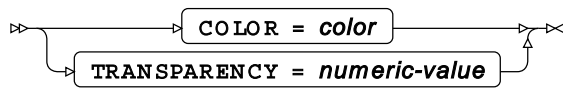
option



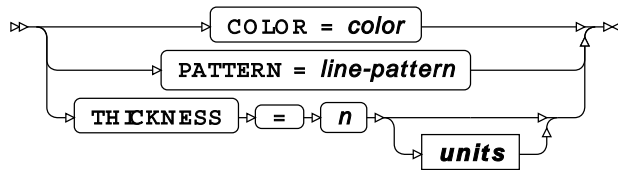
clm-option



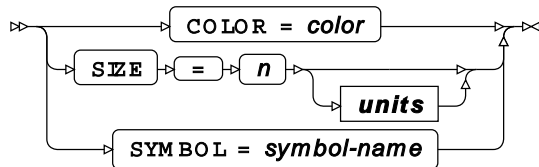
fill-option



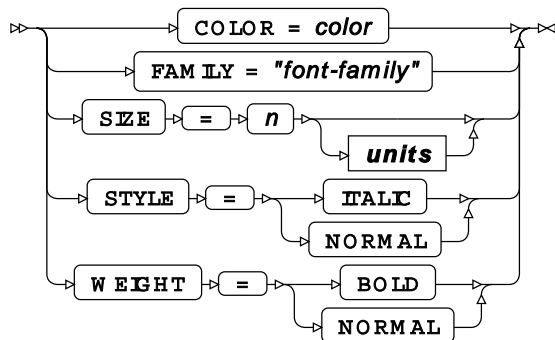
line-option



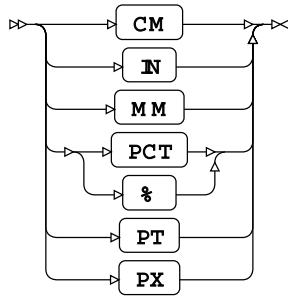
marker-option



text-option

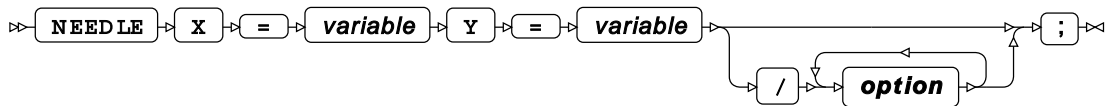


units

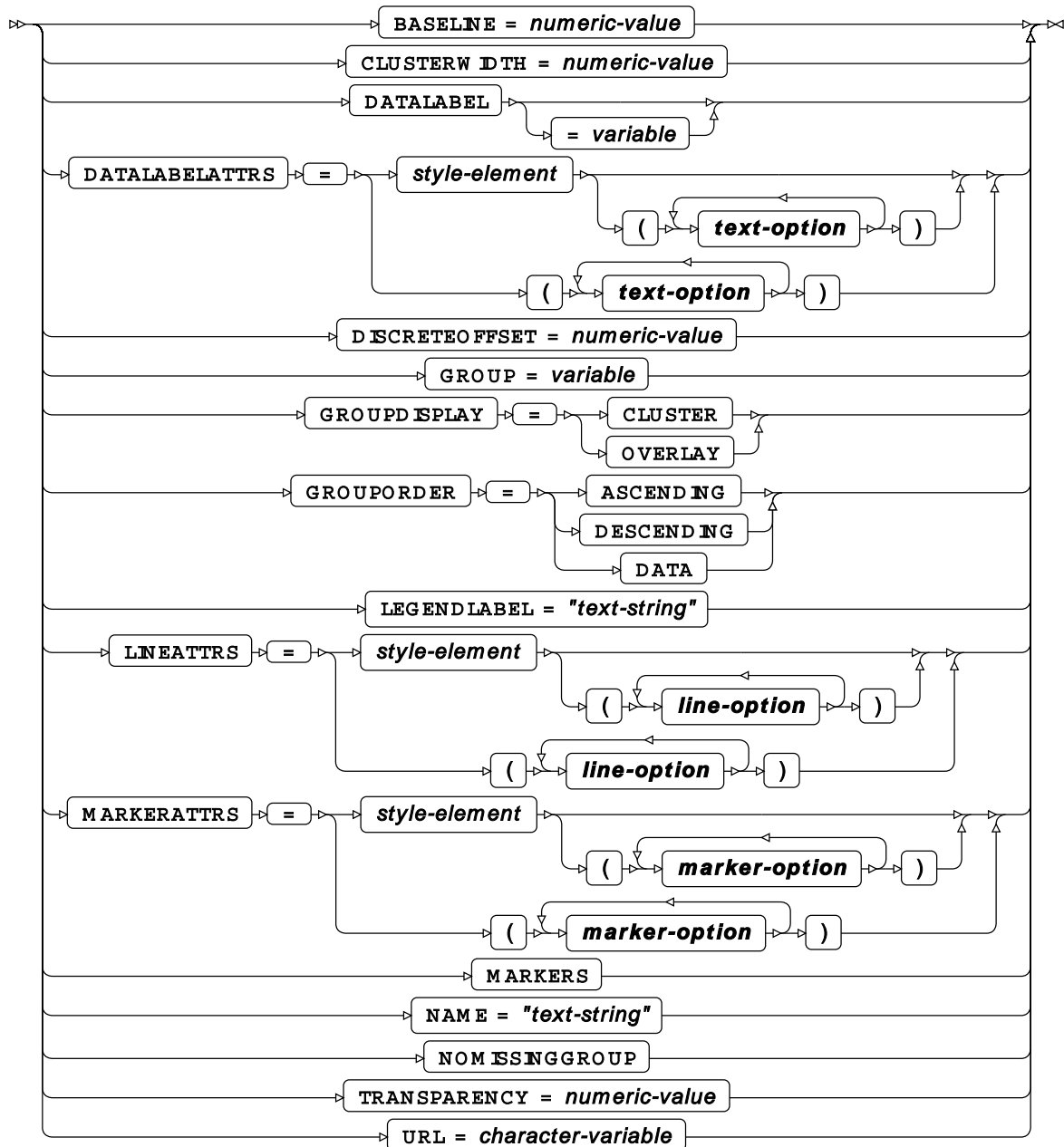


NEEDLE

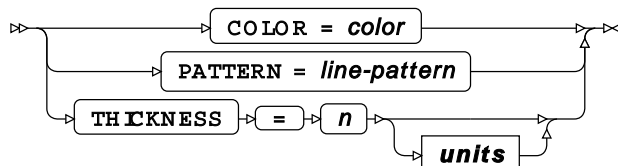
Draws needle plots.



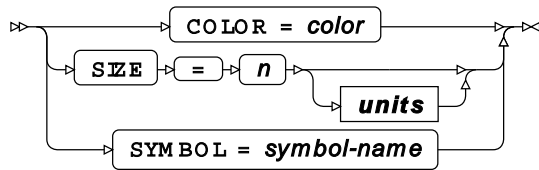
option



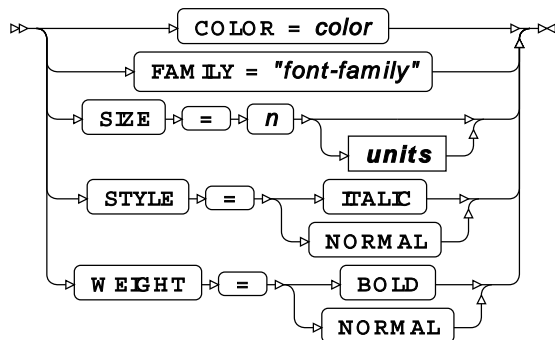
line-option



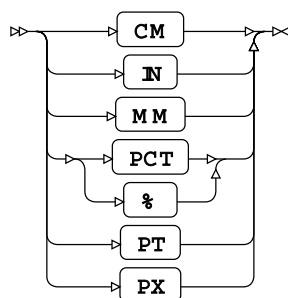
marker-option



text-option

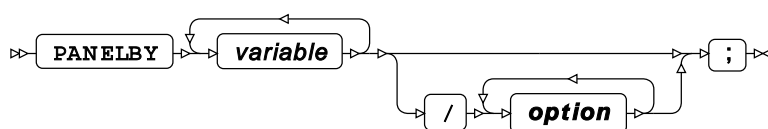


units

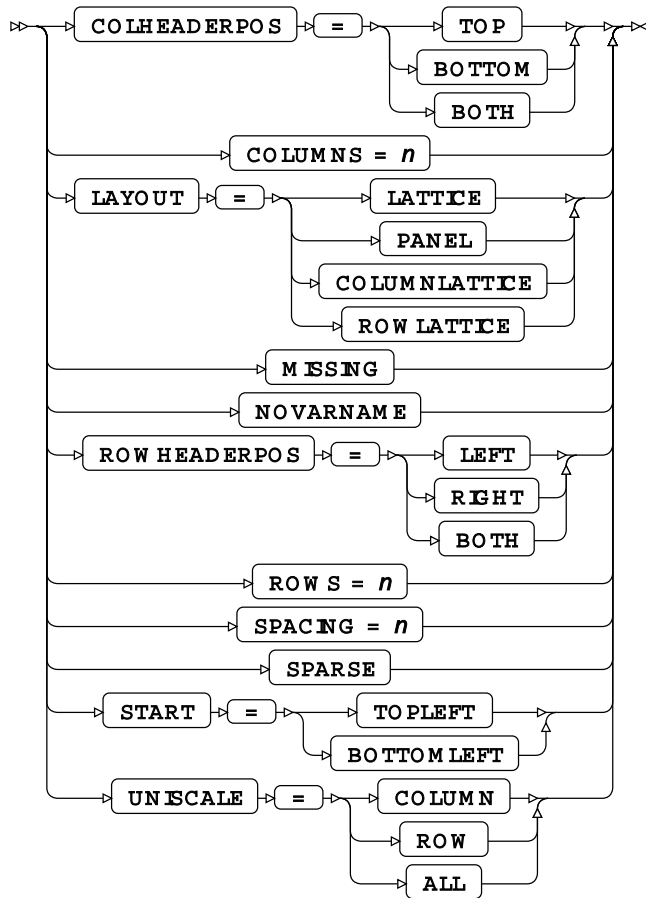


PANELBY

Used to determine the classification variables and how they are displayed.

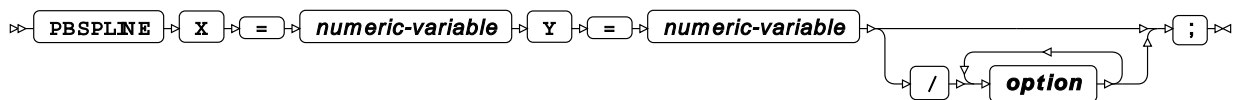


option

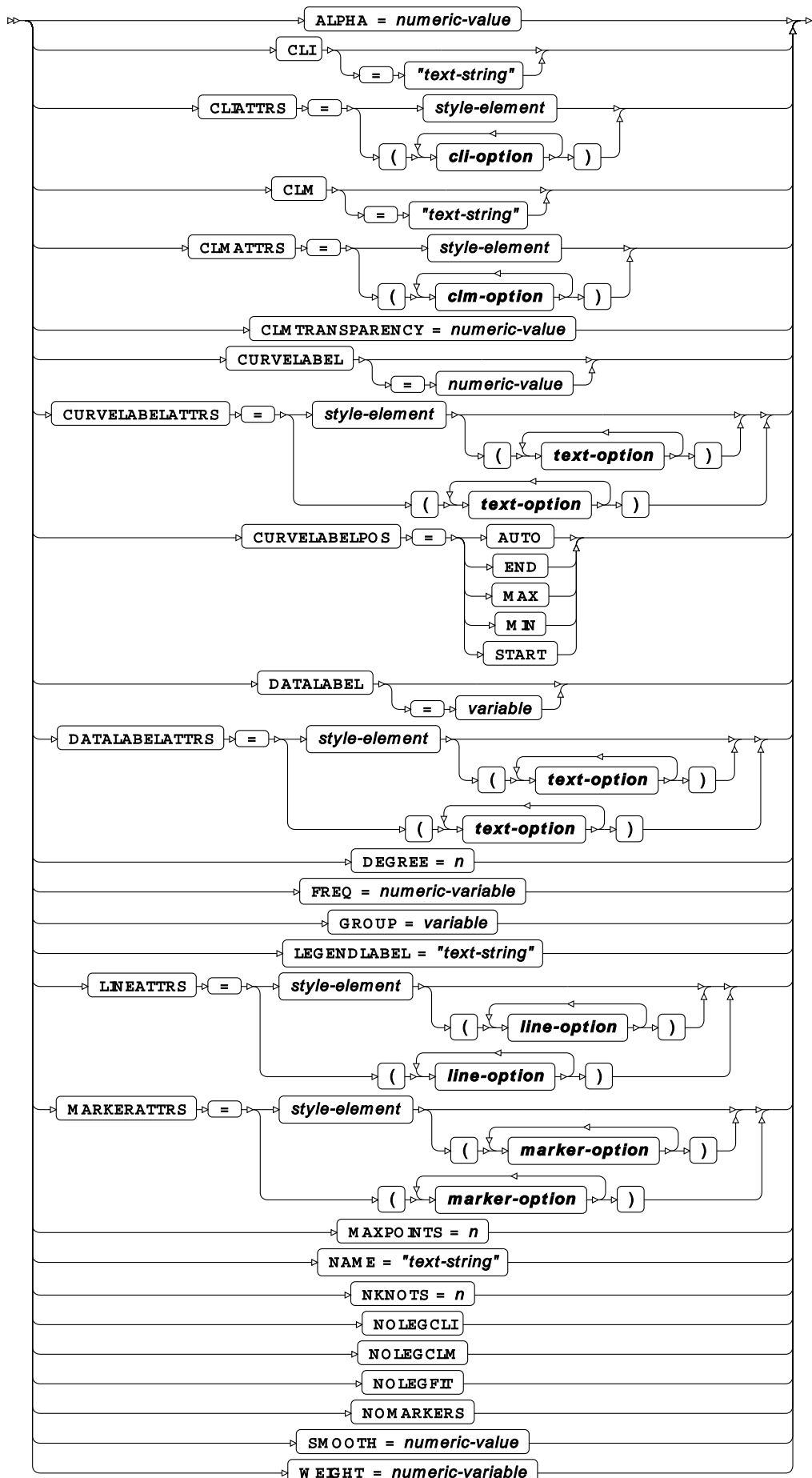


PBSPLINE

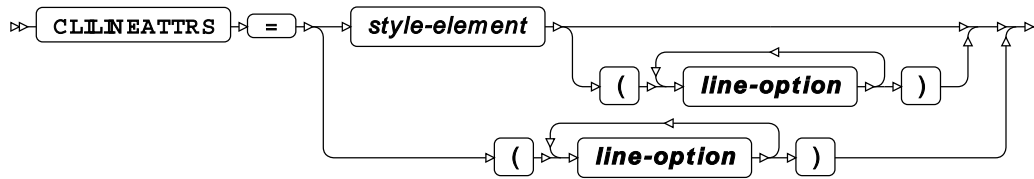
Draws fitted, penalised B-spline curves.



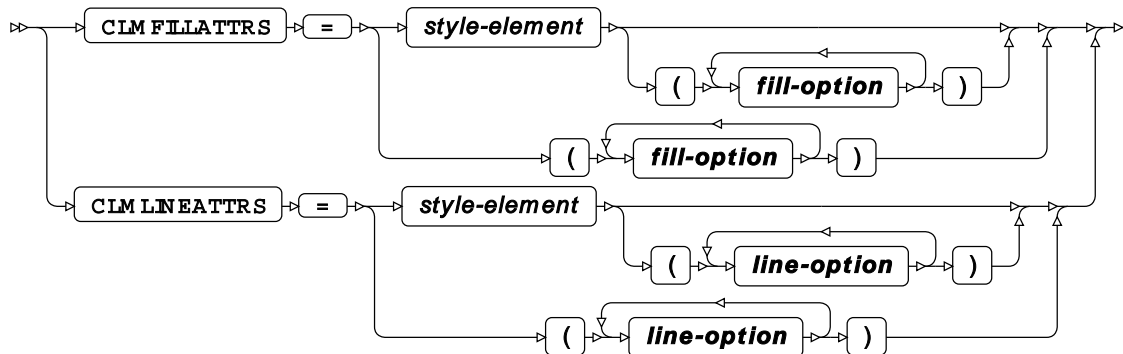
option



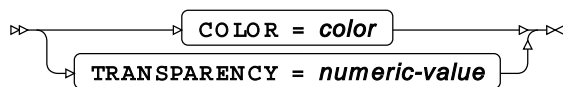
cli-option



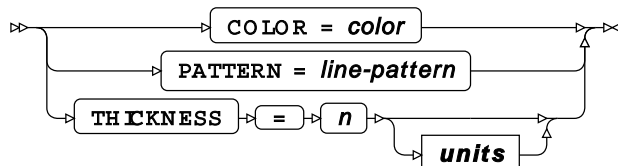
clm-option



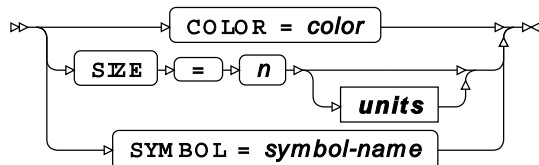
fill-option



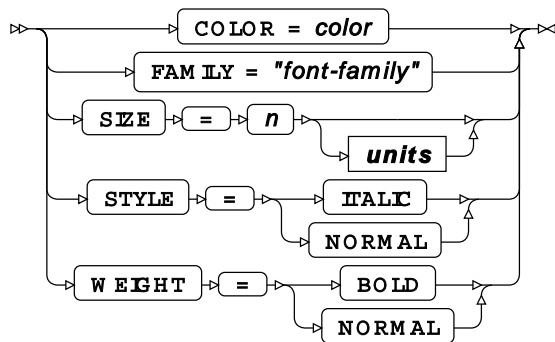
line-option



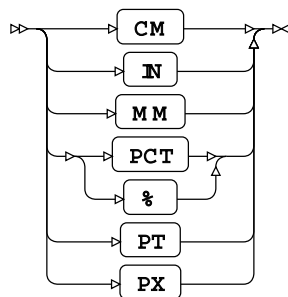
marker-option



text-option

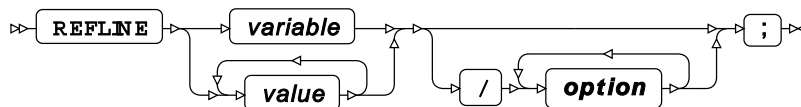


units

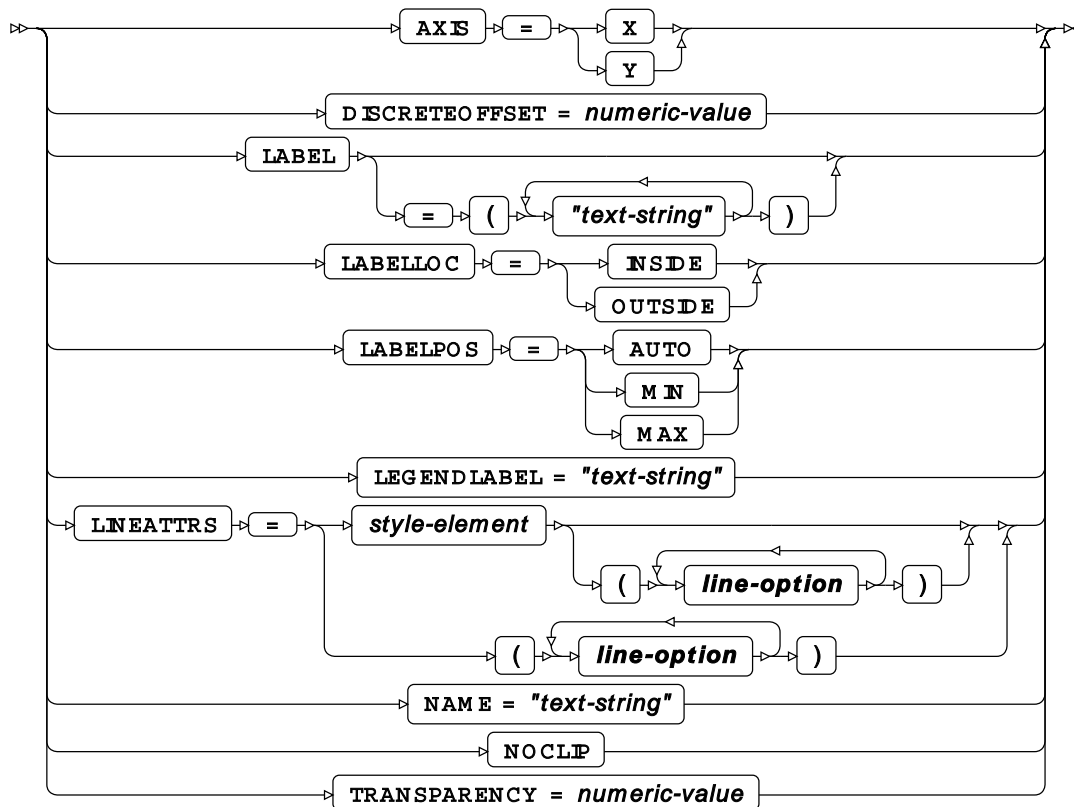


REFLINE

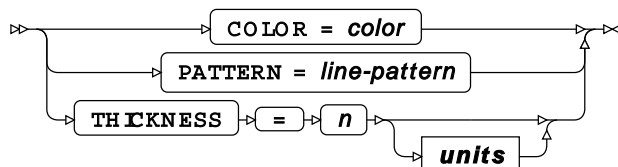
Draws one or more horizontal or vertical reference lines.



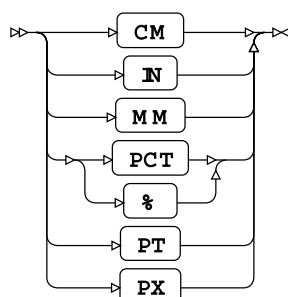
option



line-option

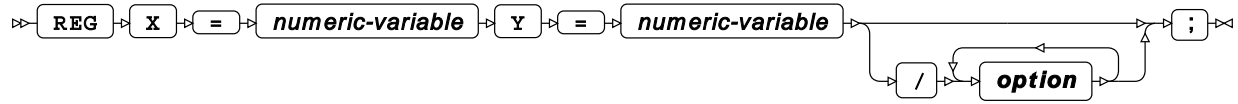


units

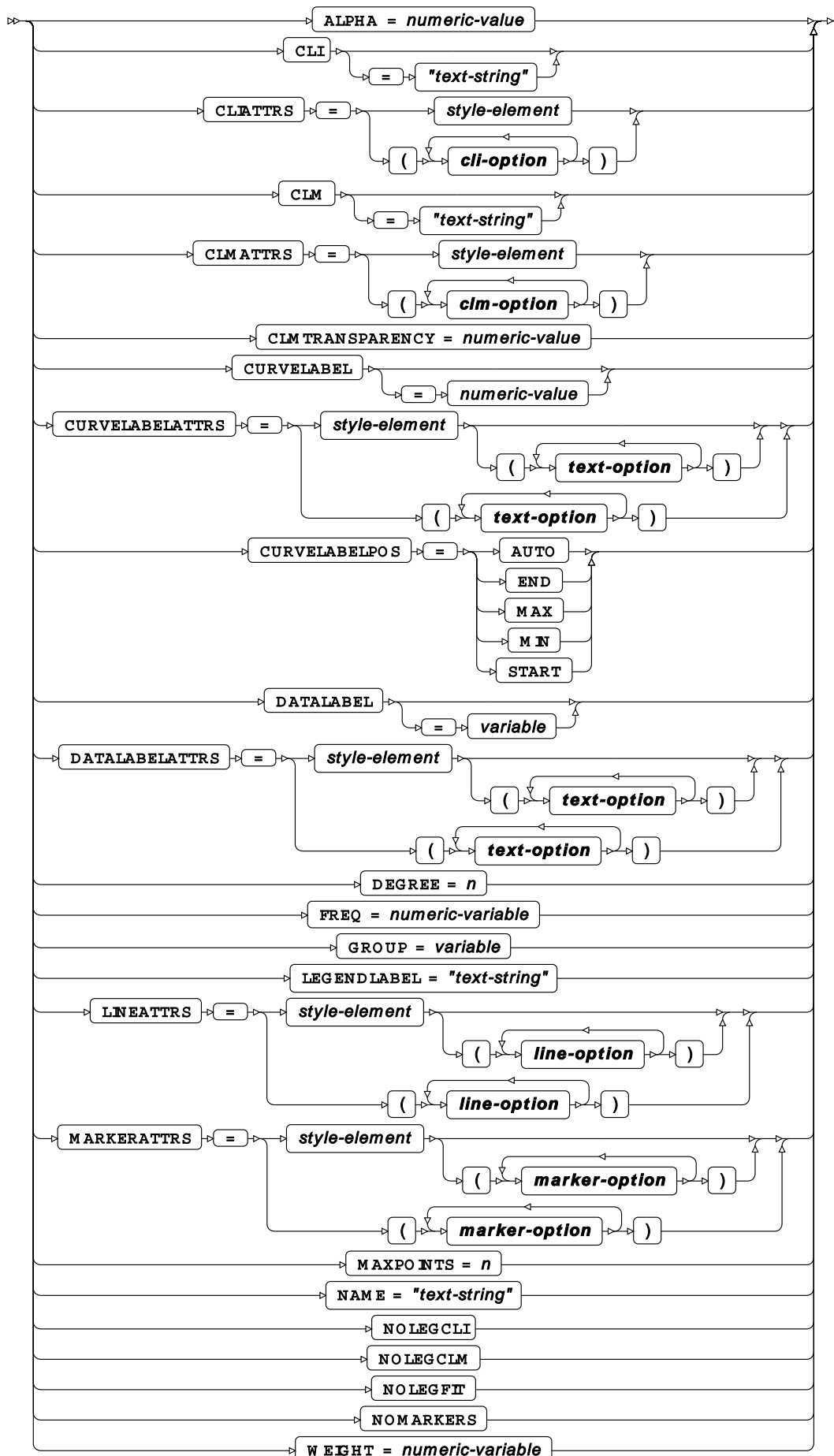


REG

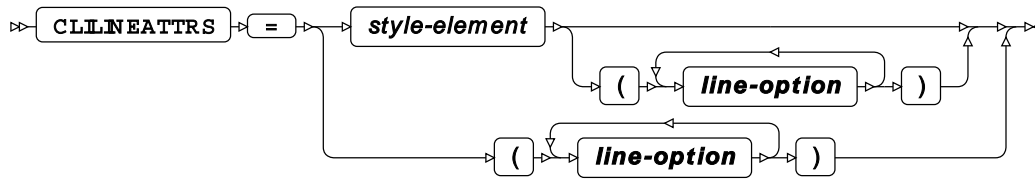
Draws fitted regression lines or curves.



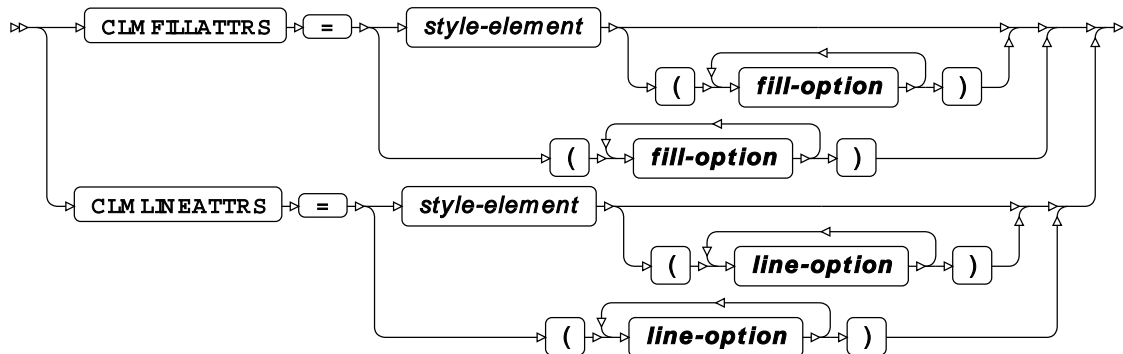
option



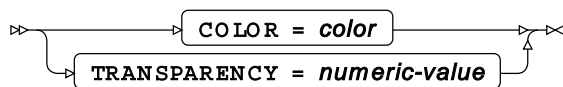
cli-option



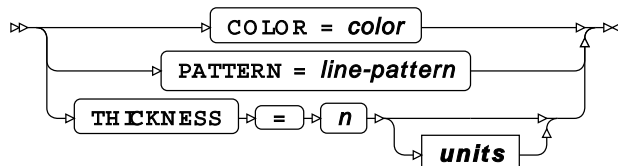
clm-option



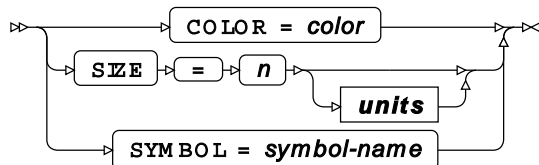
fill-option



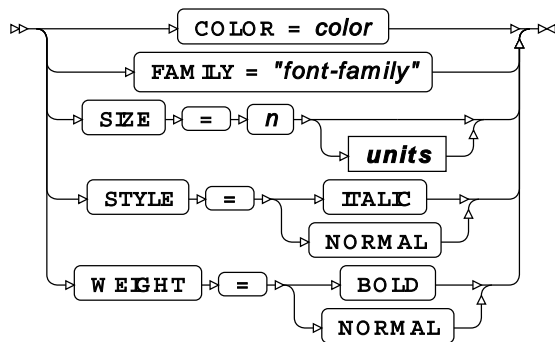
line-option



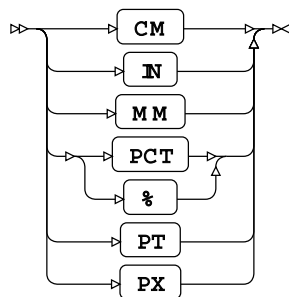
marker-option



text-option

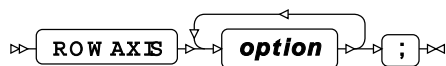


units

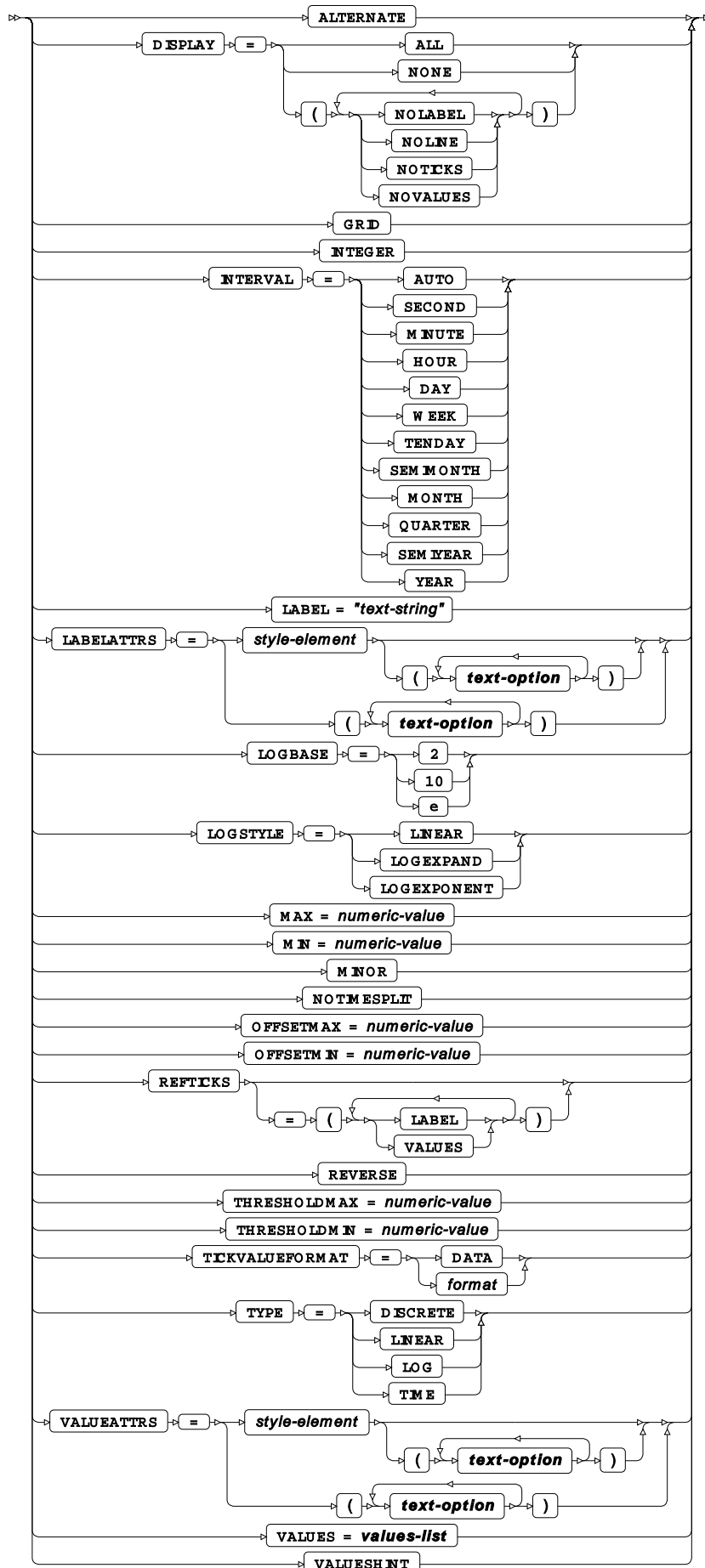


ROWAXIS

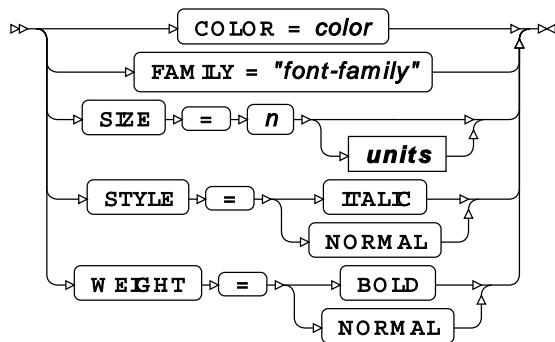
Specifies options for the row axes used in each graph



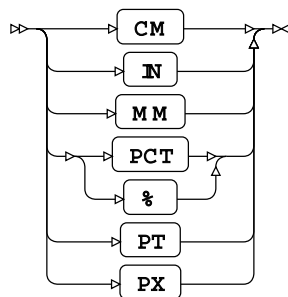
option



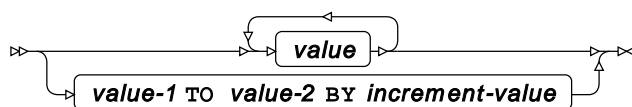
text-option



units

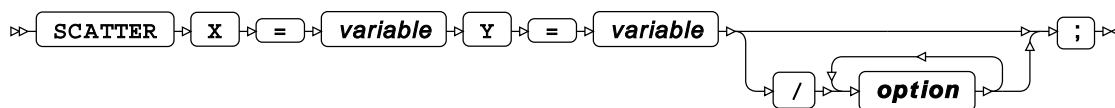


values-list

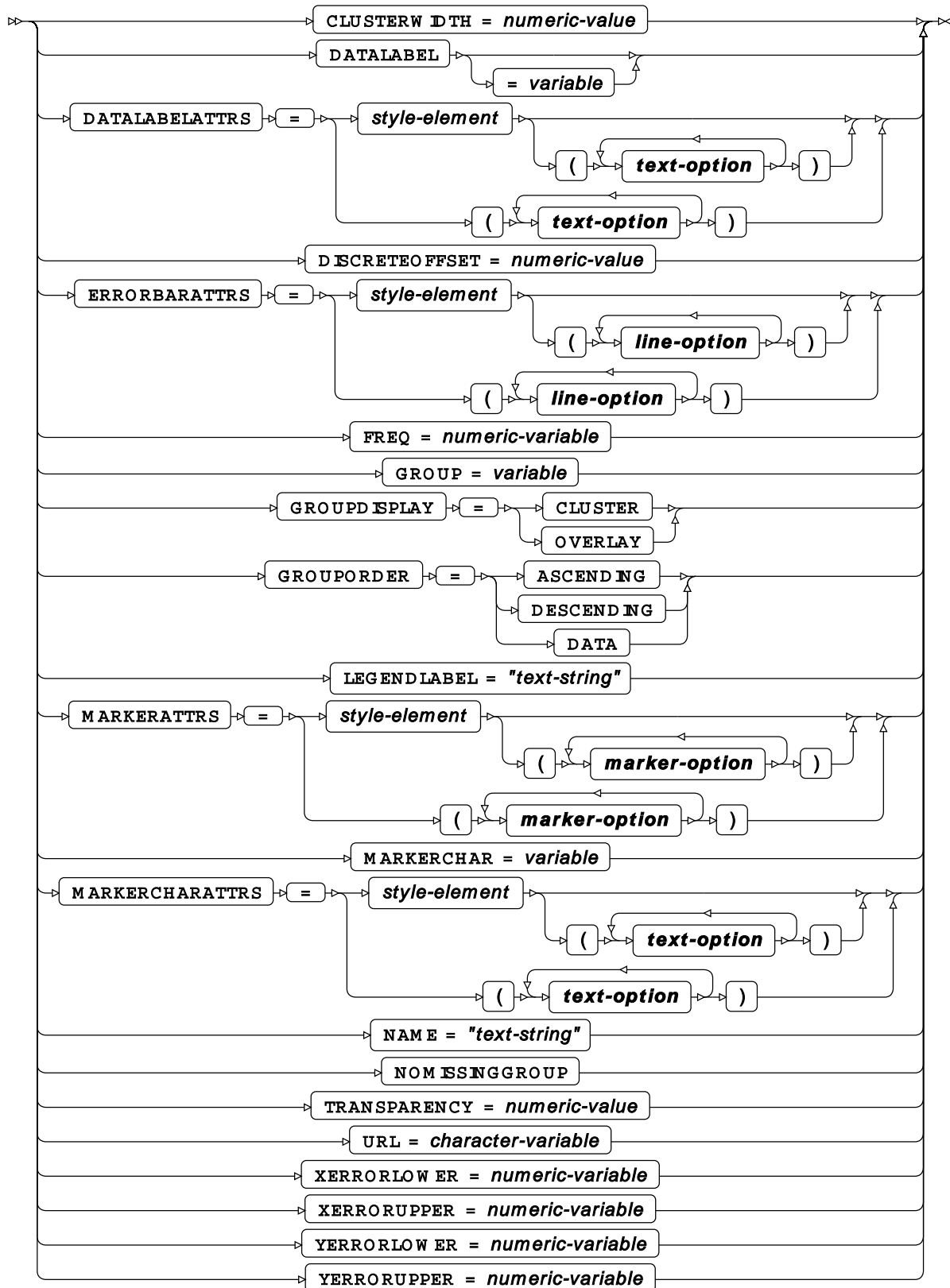


SCATTER

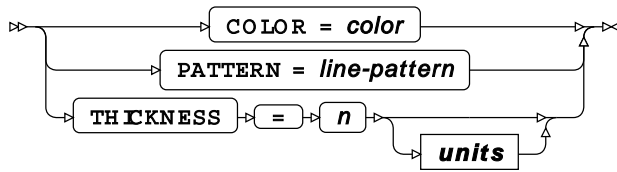
Draws scatter plots.



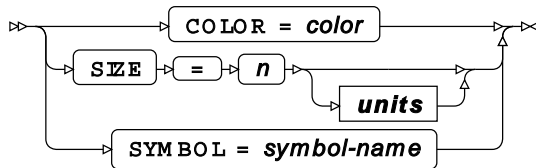
option



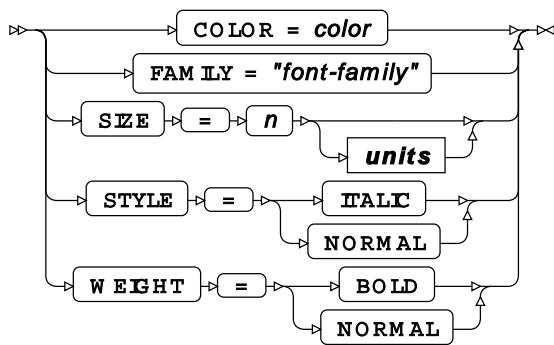
line-option



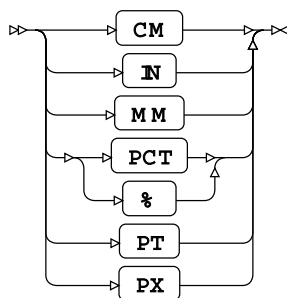
marker-option



text-option

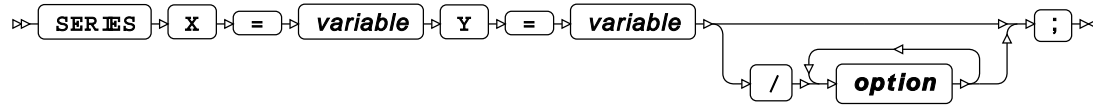


units

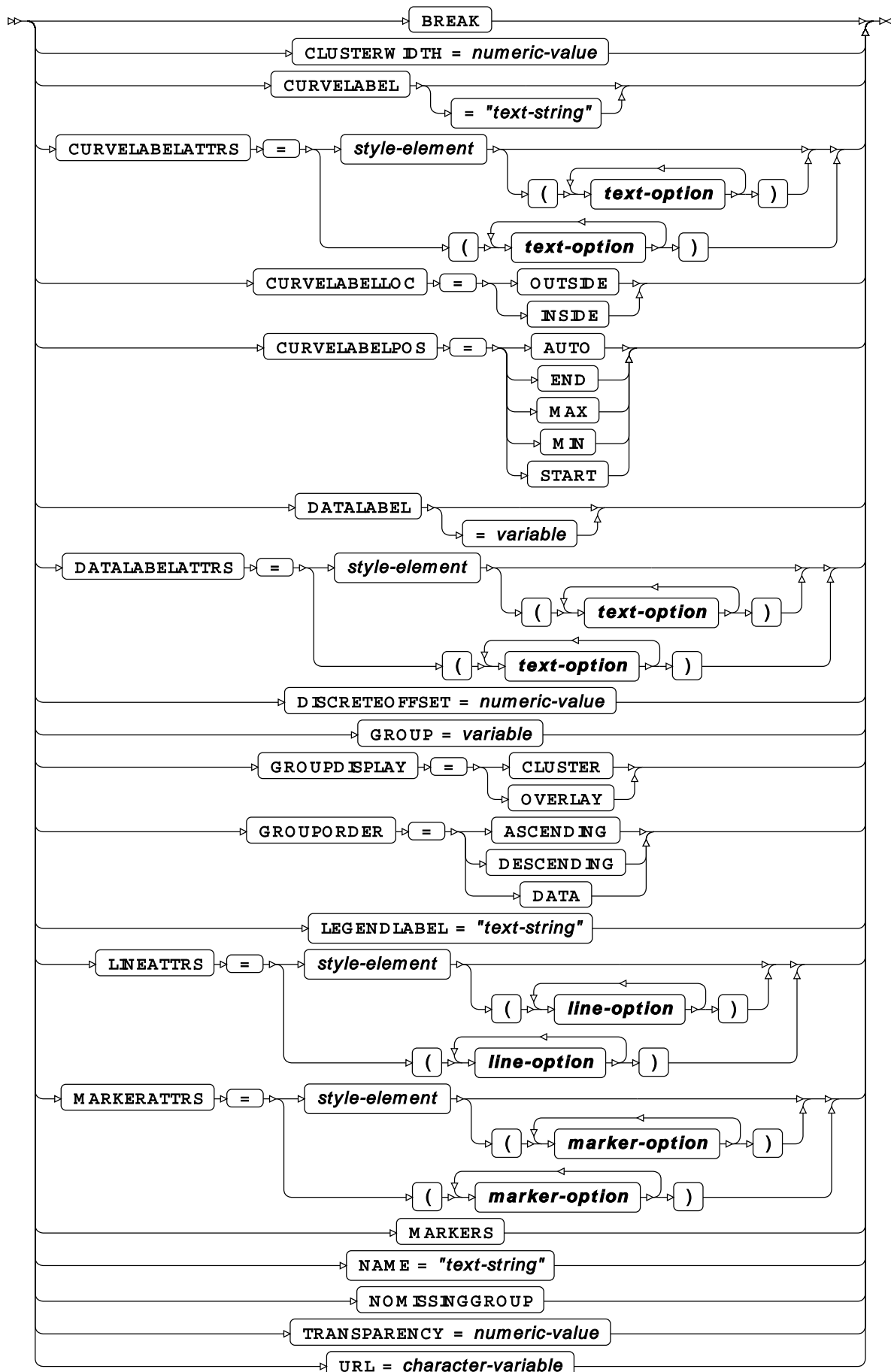


SERIES

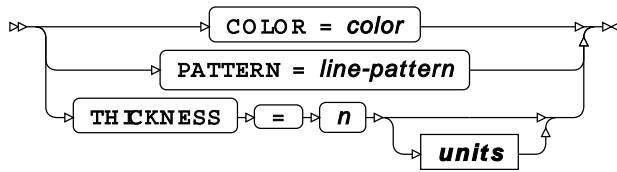
Draws series plots using unsummarised data.



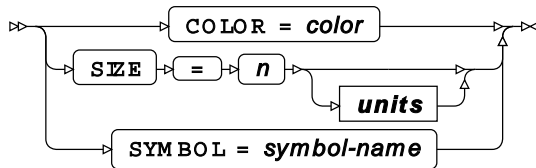
option



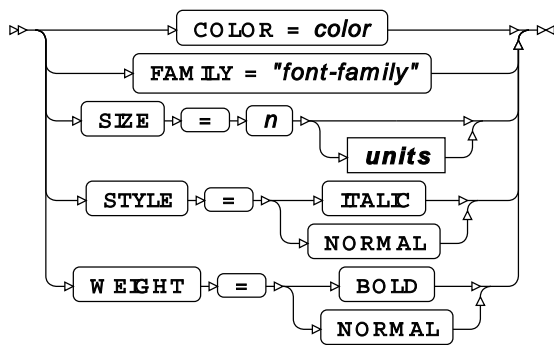
line-option



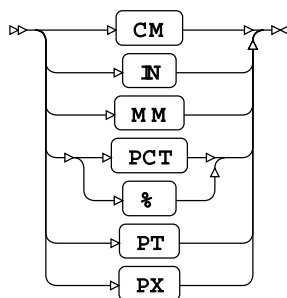
marker-option



text-option

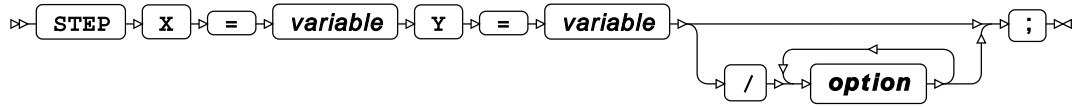


units

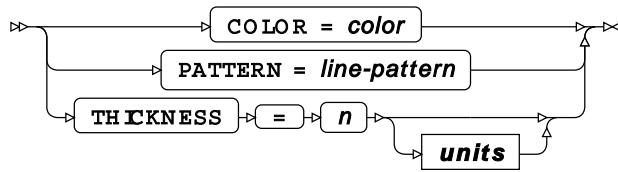


STEP

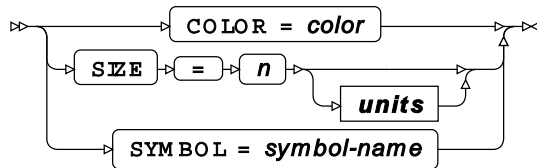
Draws step plots.



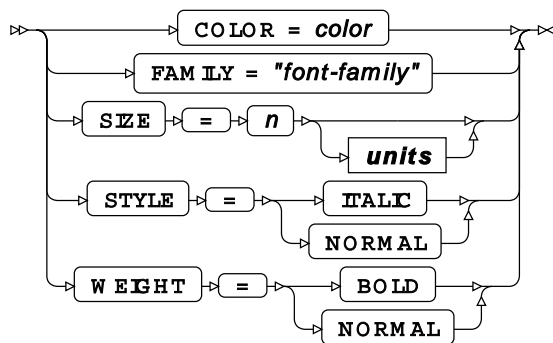
line-option



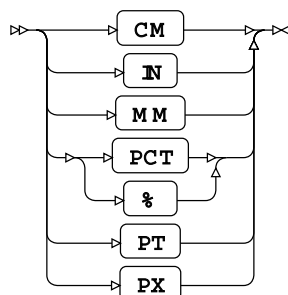
marker-option



text-option

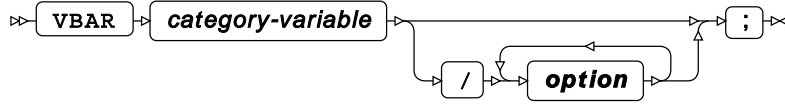


units

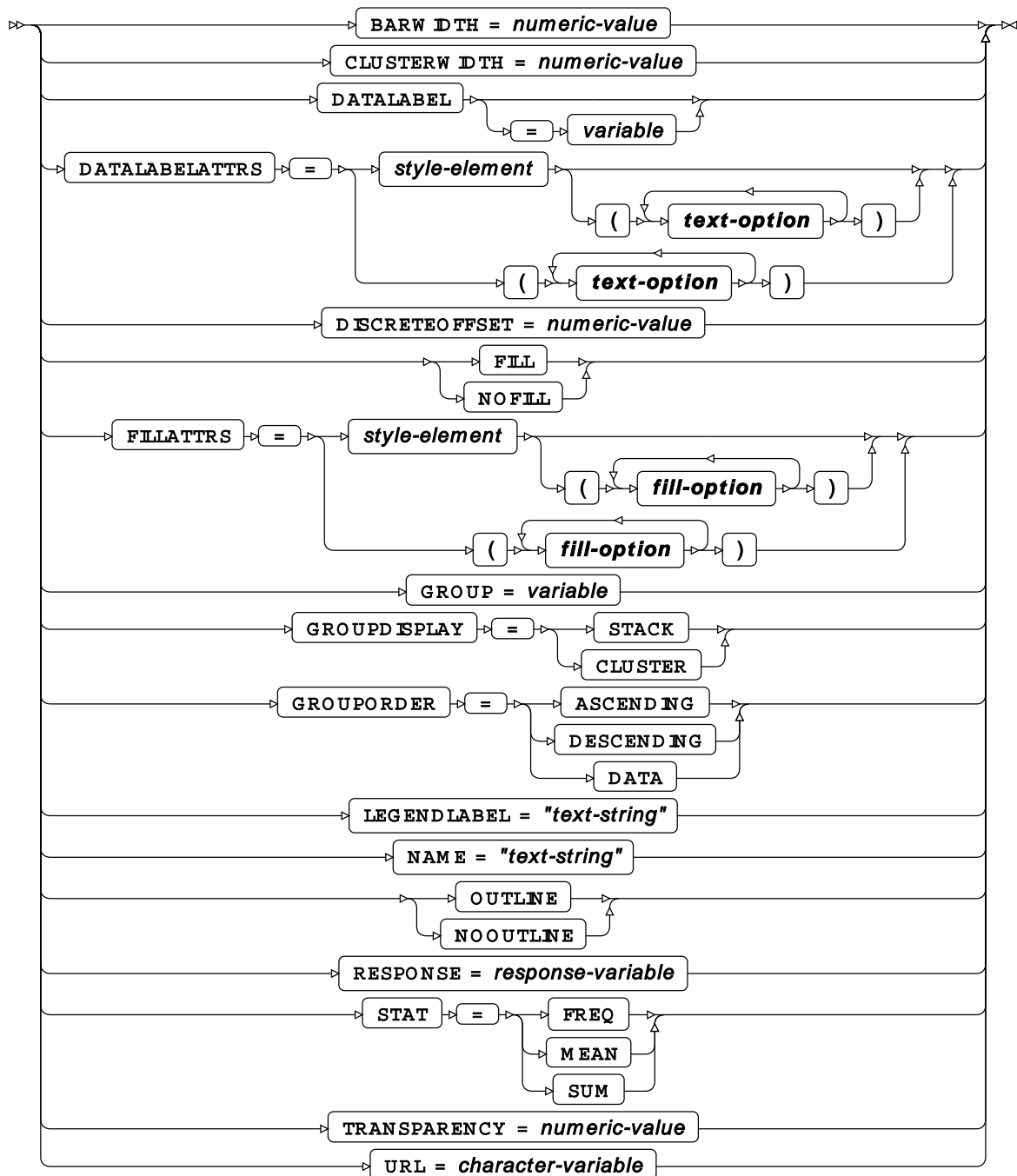


VBAR

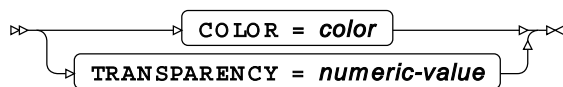
Draws vertical bar charts using unsummarised data.



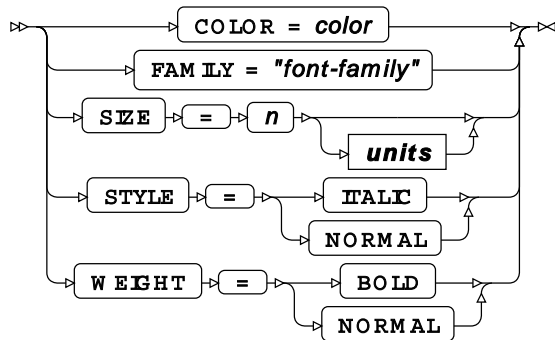
option



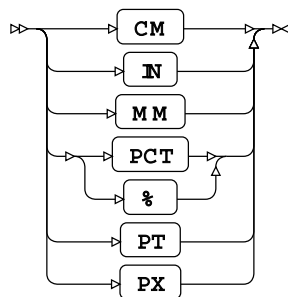
fill-option



text-option

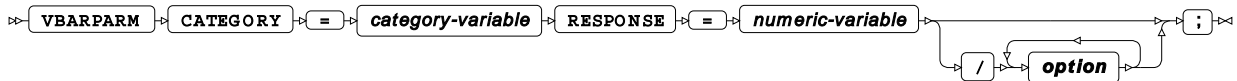


units

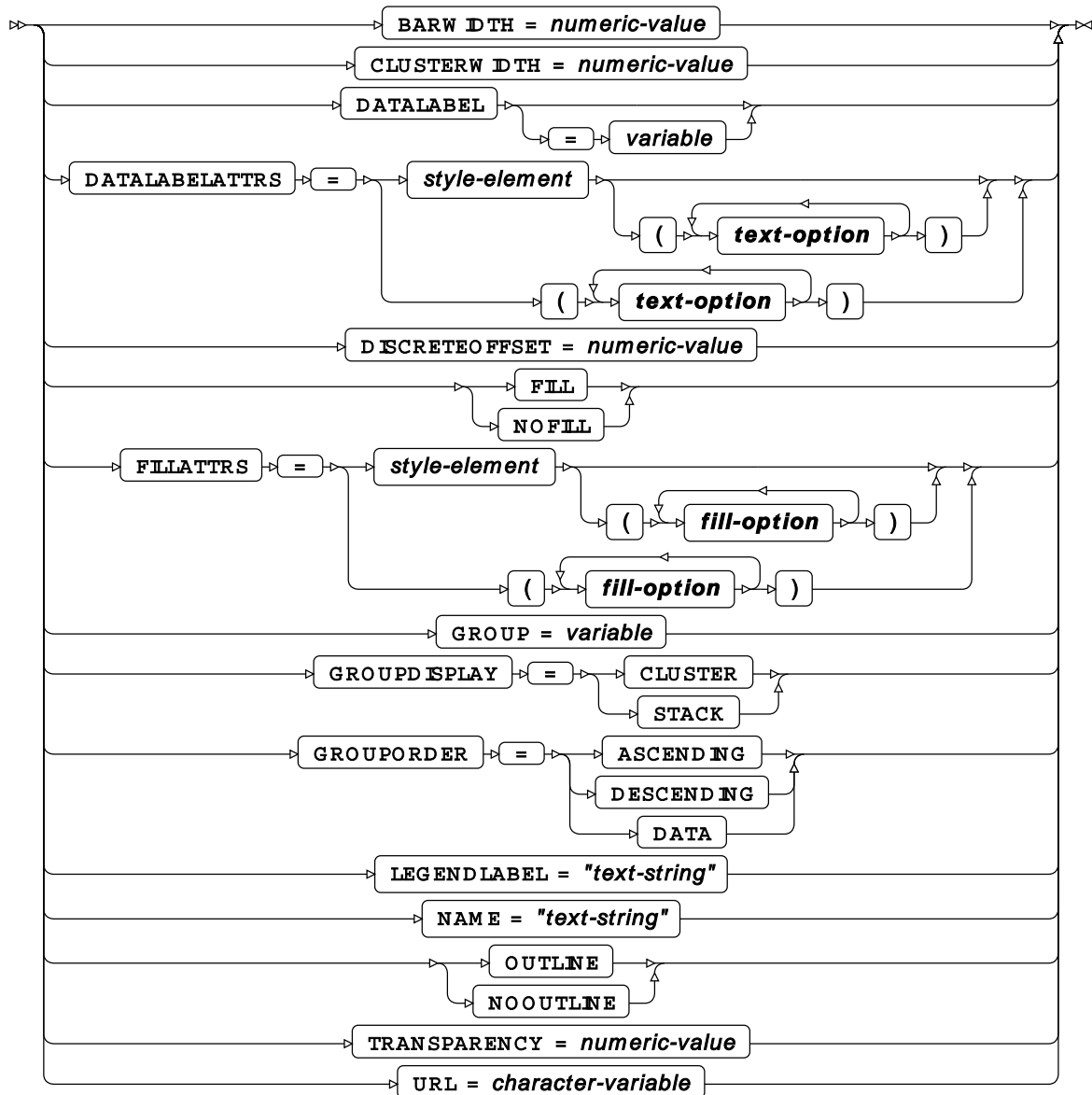


VBARPARM

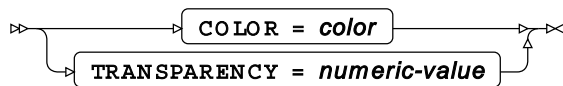
Draws vertical bar charts using summarised data.



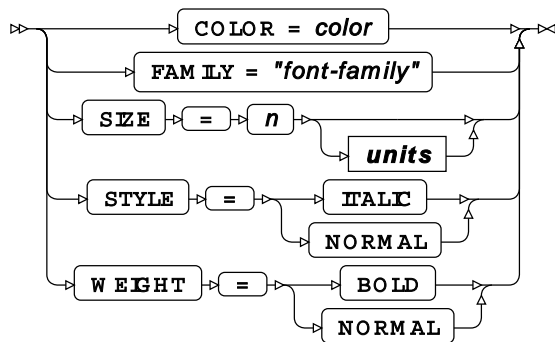
option



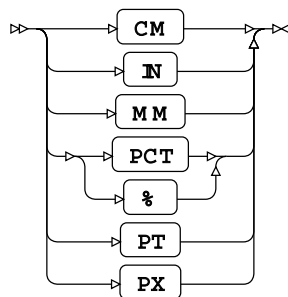
fill-option



text-option

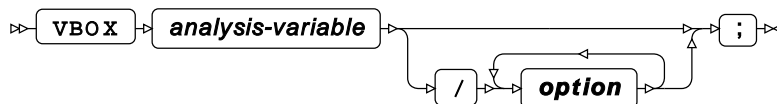


units

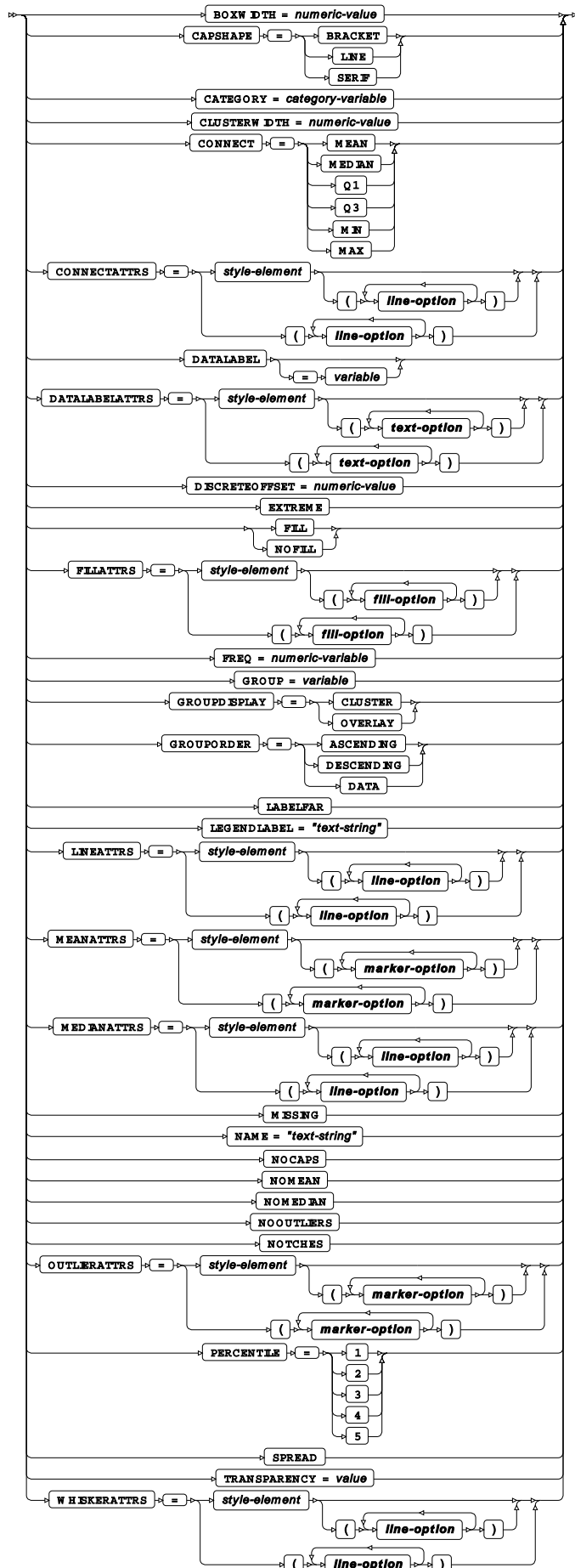


VBOX

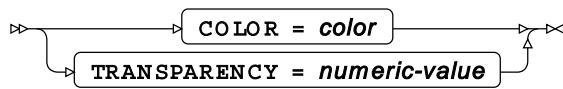
Draws vertical box plots.



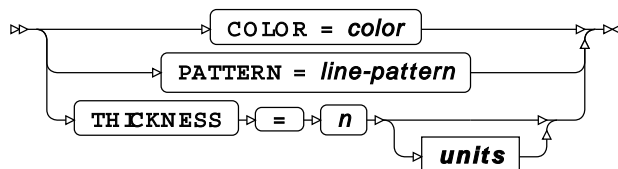
option



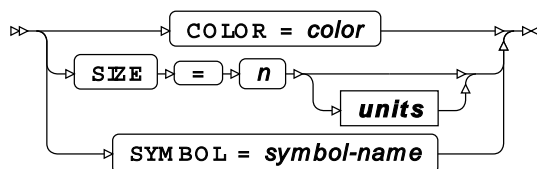
fill-option



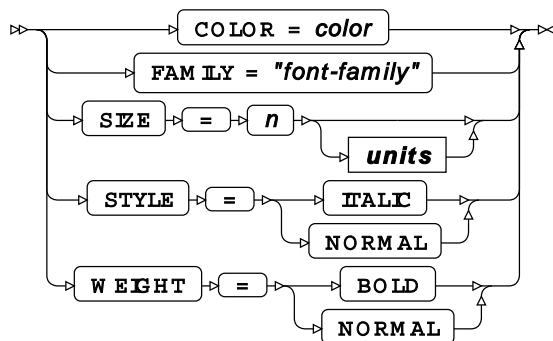
line-option



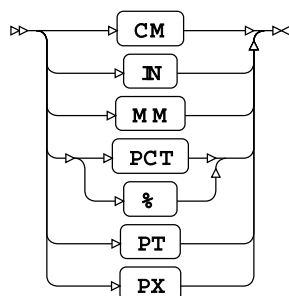
marker-option



text-option

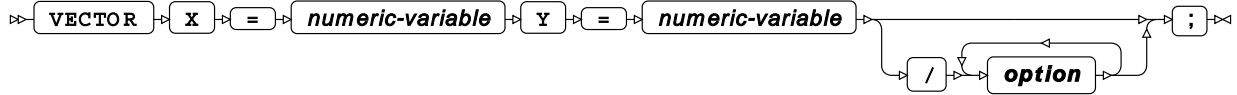


units

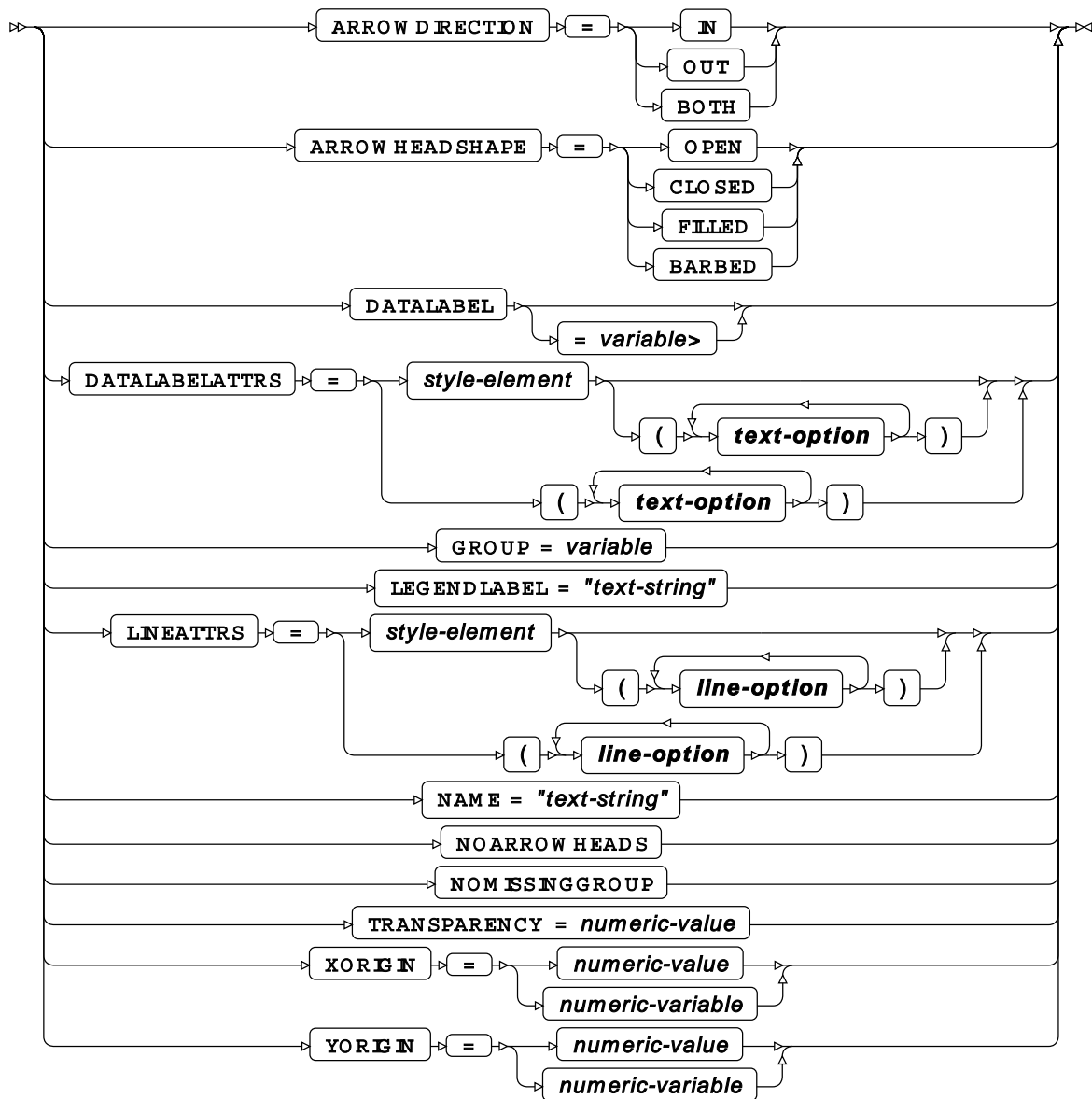


VECTOR

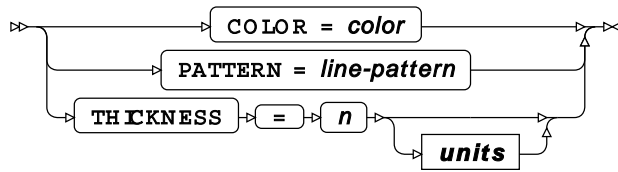
Draws vector plots.



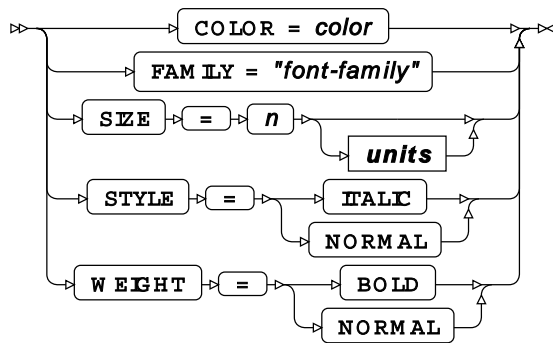
option



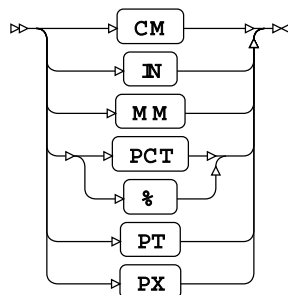
line-option



text-option

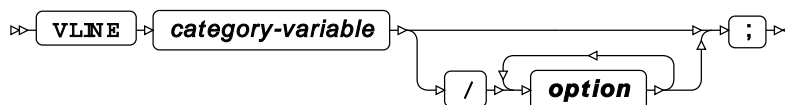


units

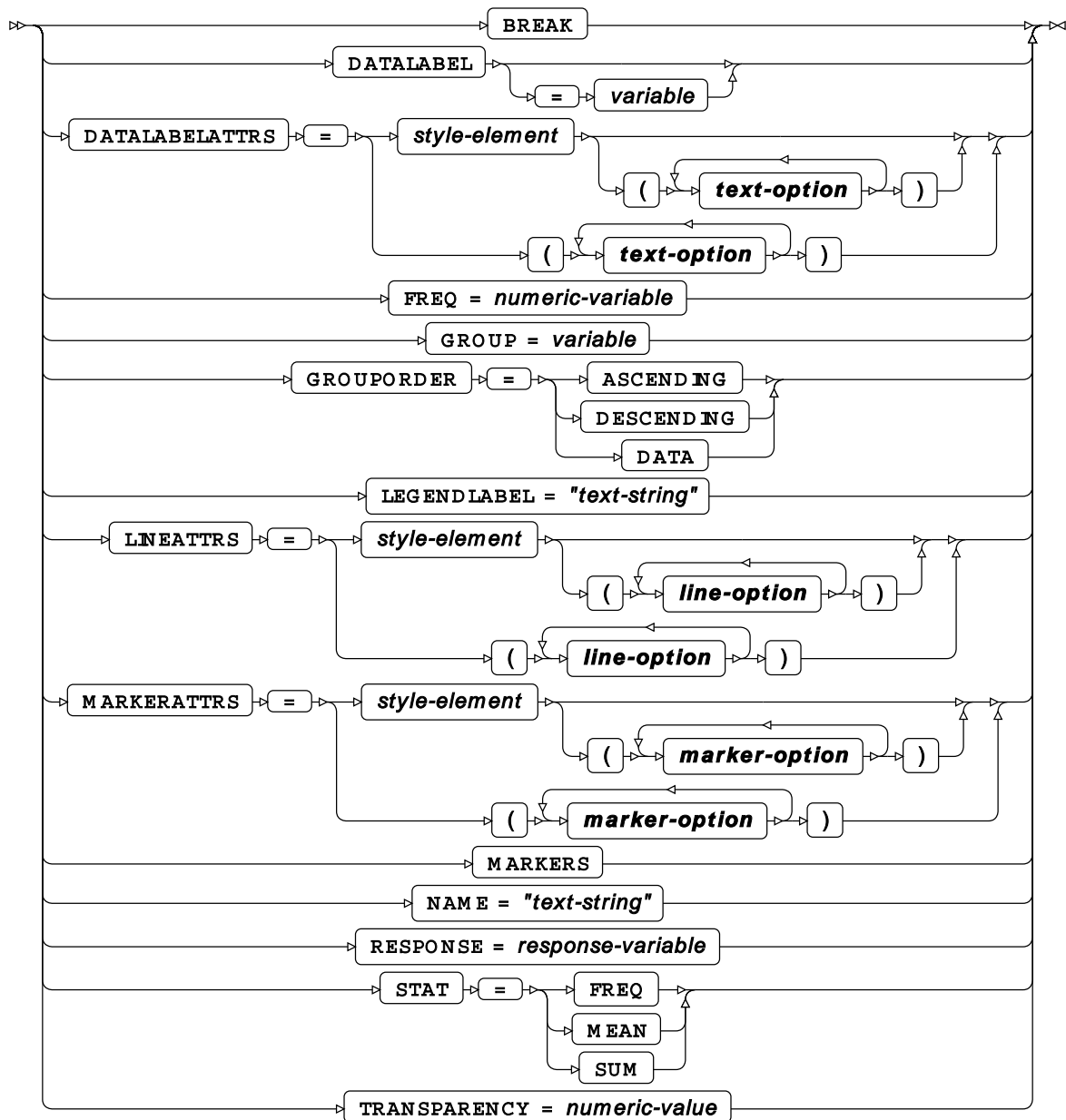


VLINE

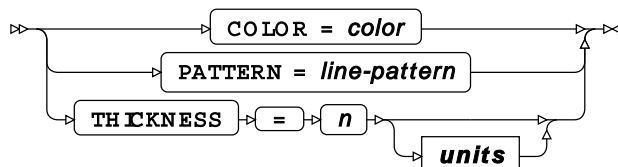
Draws vertical line plots using unsummarised data.



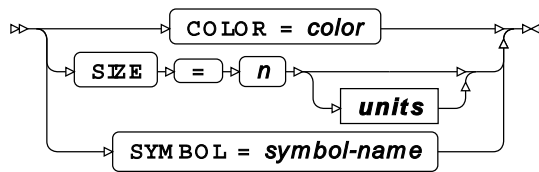
option



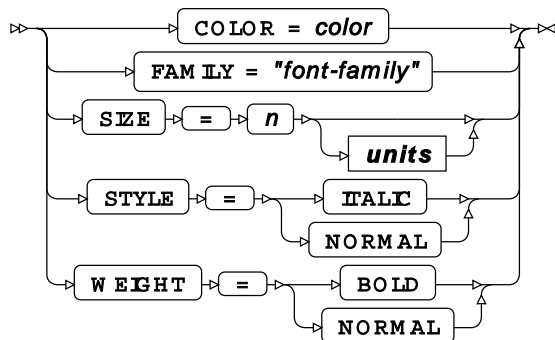
line-option



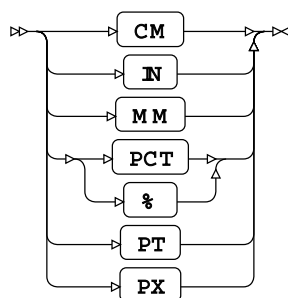
marker-option



text-option

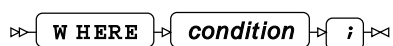


units



WHERE

Restricts the observations to be processed.



SGPLOT procedure

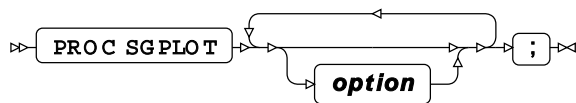
Supported statements

- *PROC SGPLOT* [↗](#) (page 2710)
- *BAND* [↗](#) (page 2711)
- *BUBBLE* [↗](#) (page 2713)
- *BY* [↗](#) (page 2715)
- *DENSITY* [↗](#) (page 2716)
- *ELLIPSE* [↗](#) (page 2717)
- *FORMAT* [↗](#) (page 2719)
- *HBAR* [↗](#) (page 2719)
- *HBARPARM* [↗](#) (page 2721)
- *HBOX* [↗](#) (page 2723)
- *HLINE* [↗](#) (page 2729)
- *HIGHLOW* [↗](#) (page 2726)
- *HISTOGRAM* [↗](#) (page 2728)
- *KEYLEGEND* [↗](#) (page 2731)
- *LABEL* [↗](#) (page 2733)
- *LINEPARM* [↗](#) (page 2733)
- *LOESS* [↗](#) (page 2735)
- *NEEDLE* [↗](#) (page 2738)
- *PBSPLINE* [↗](#) (page 2740)
- *REFLINE* [↗](#) (page 2743)
- *REG* [↗](#) (page 2745)
- *SCATTER* [↗](#) (page 2748)
- *SERIES* [↗](#) (page 2751)
- *STEP* [↗](#) (page 2754)
- *VBAR* [↗](#) (page 2757)
- *VBARPARM* [↗](#) (page 2759)
- *VBOX* [↗](#) (page 2761)
- *VECTOR* [↗](#) (page 2764)
- *VLINE* [↗](#) (page 2765)
- *WATERFALL* [↗](#) (page 2767)
- *WHERE* [↗](#) (page 2769)

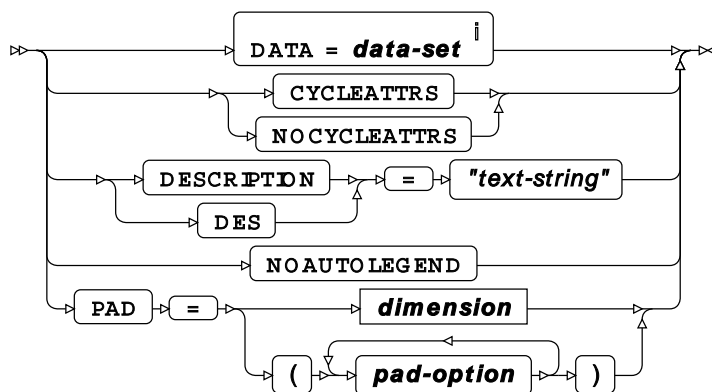
- [XAXIS](#) (page 2769)
- [X2AXIS](#) (page 2771)
- [YAXIS](#) (page 2773)
- [Y2AXIS](#) (page 2775)

PROC SGPLOT

Draws one or more plots on a single set of axes

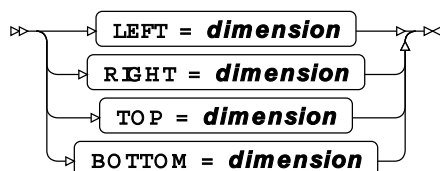


option

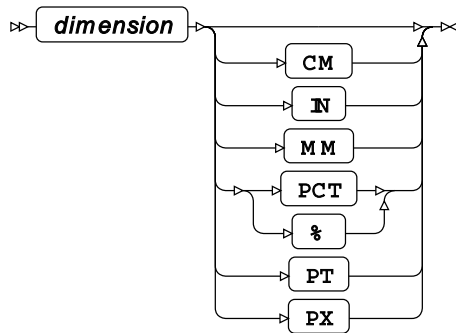


ⁱ See [Input dataset](#) (page 16).

pad-option

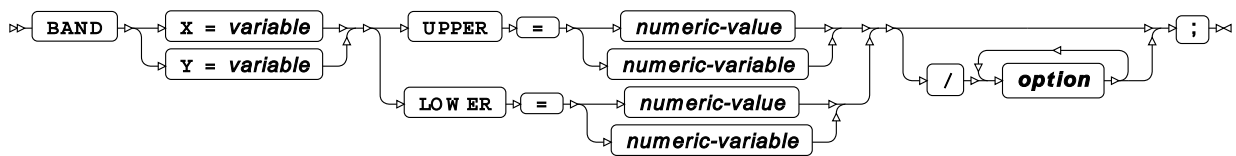


dimension

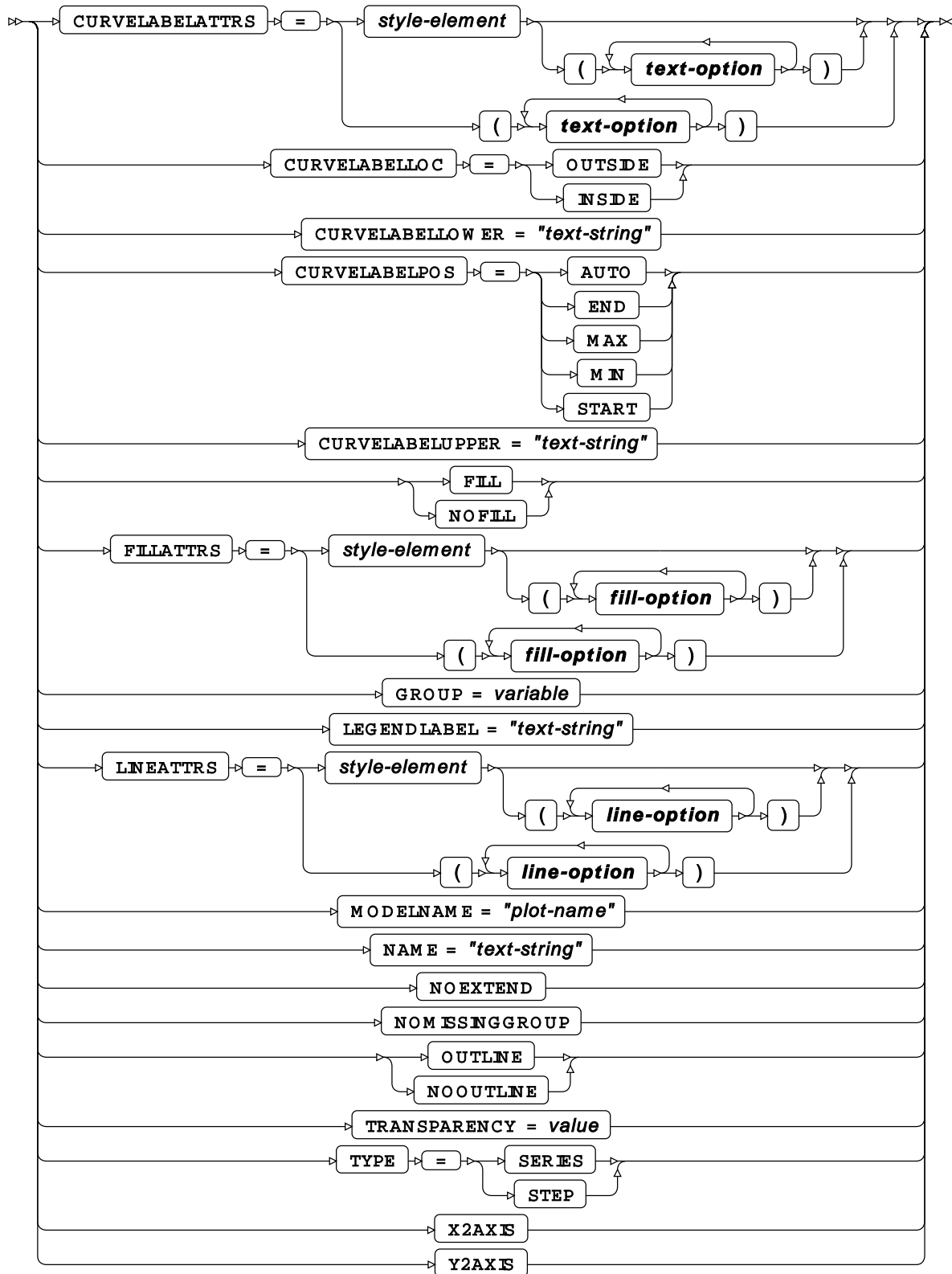


BAND

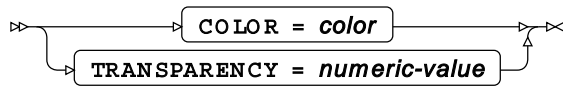
Draws a band plot.



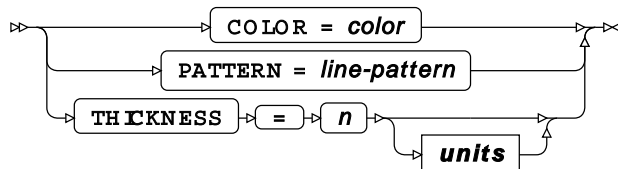
option



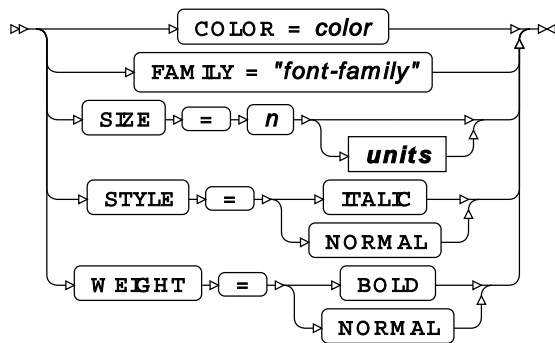
fill-option



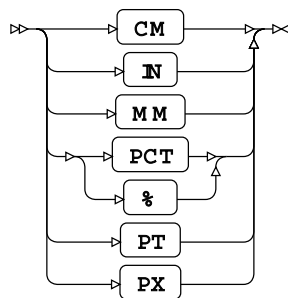
line-option



text-option

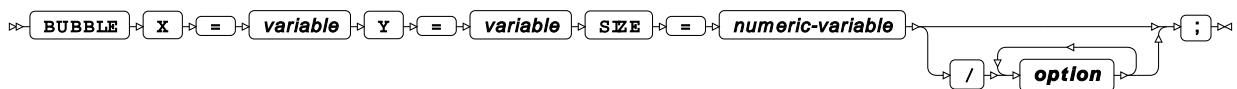


units

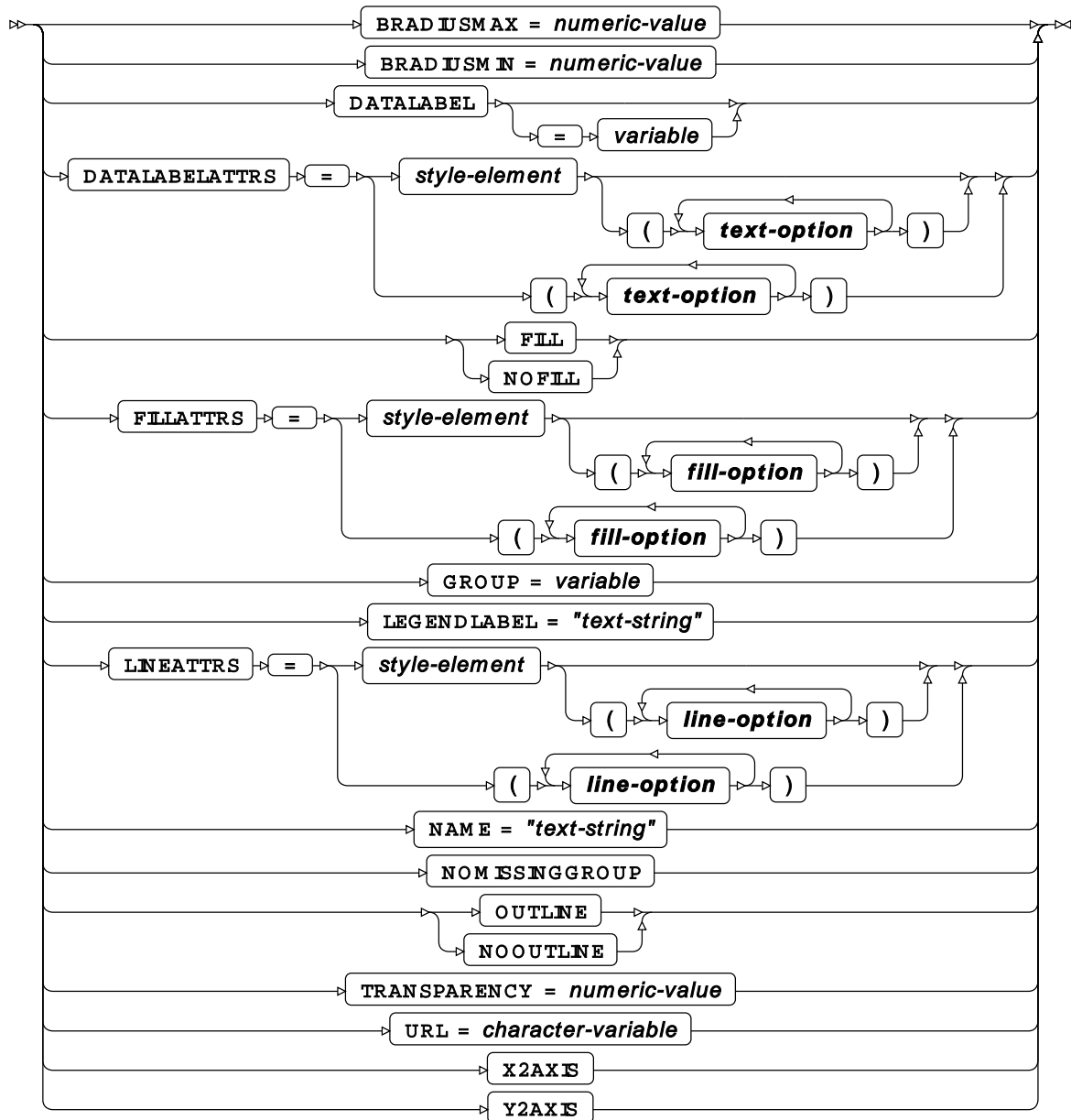


BUBBLE

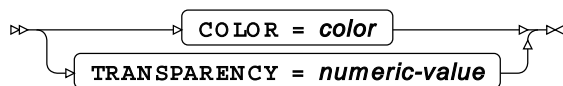
Draws a bubble plot.



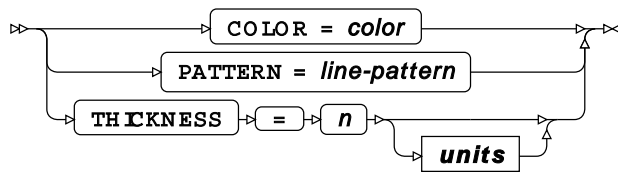
option



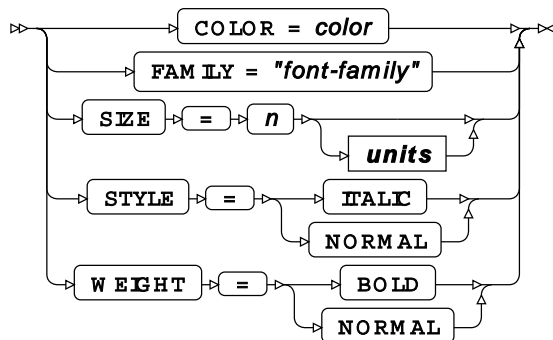
fill-option



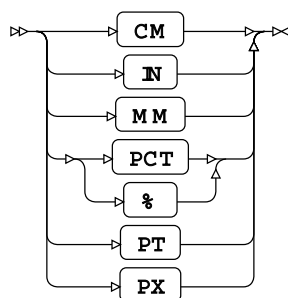
line-option



text-option

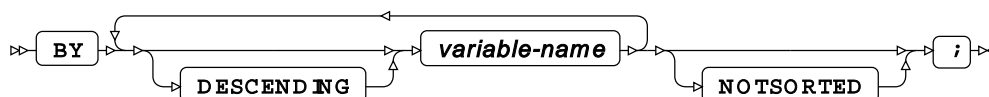


units



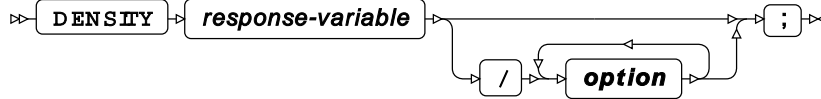
BY

Groups the observations in a dataset using one or more specified variables.

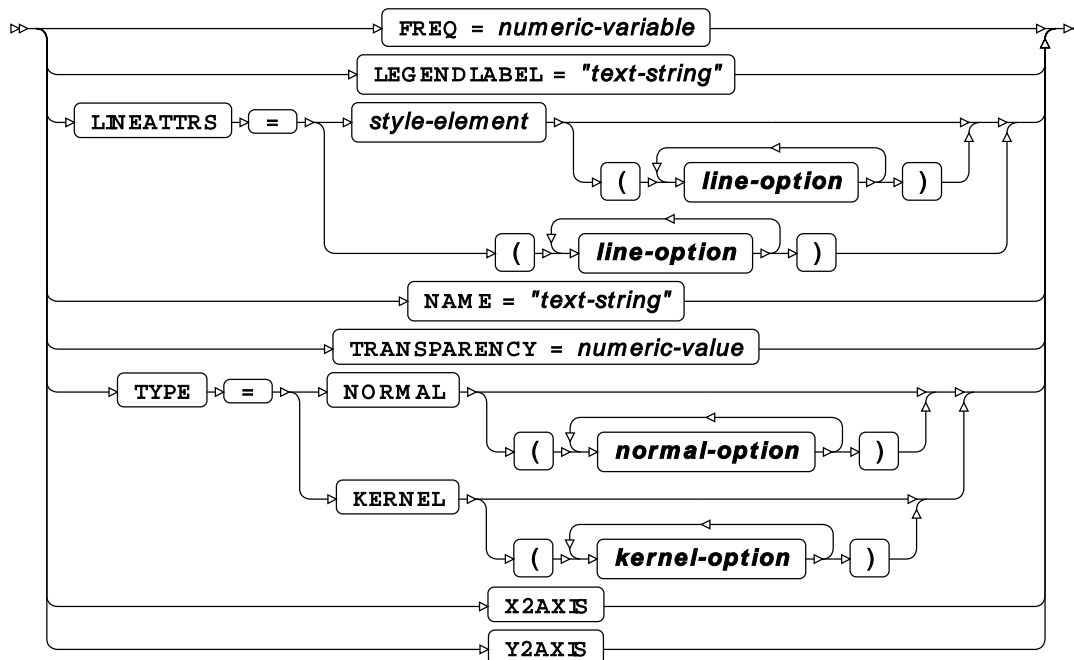


DENSITY

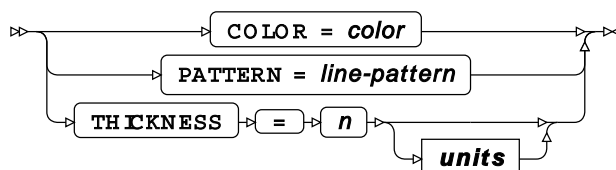
Draws a density curve.



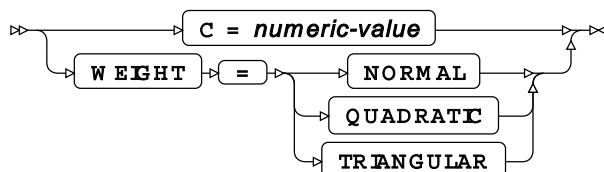
option



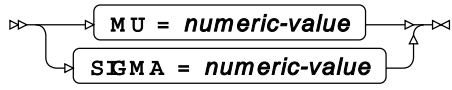
line-option



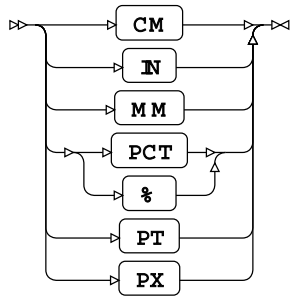
kernel-option



normal-option

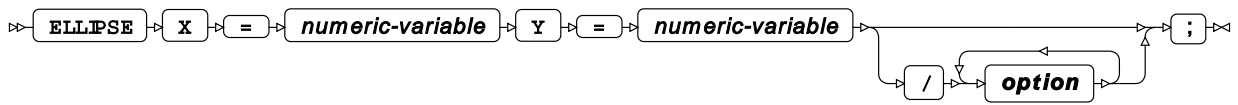


units

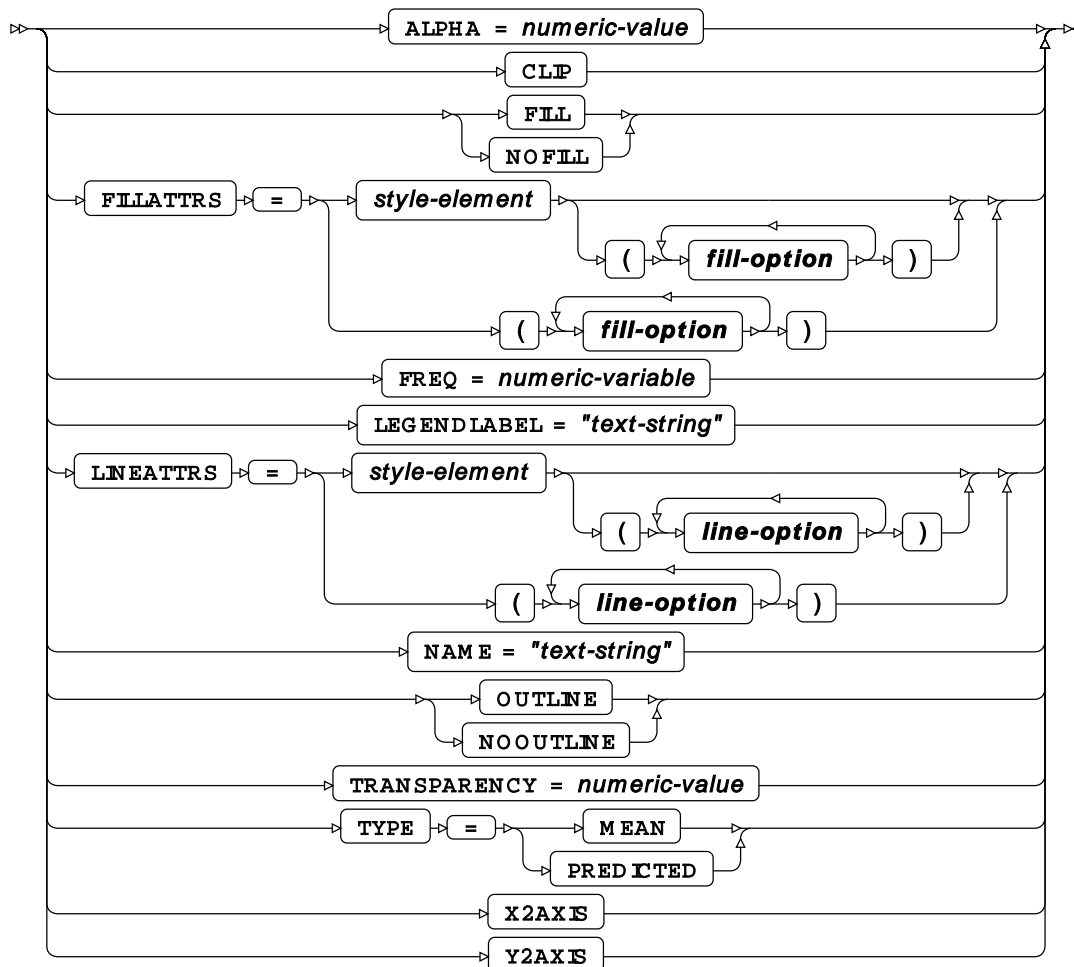


ELLIPSE

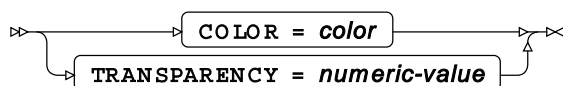
Draws a confidence or prediction ellipse.



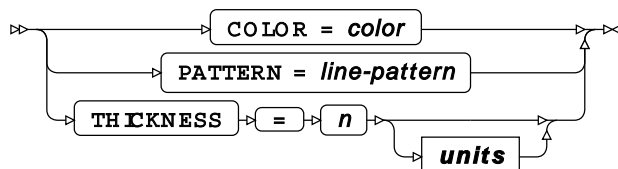
option



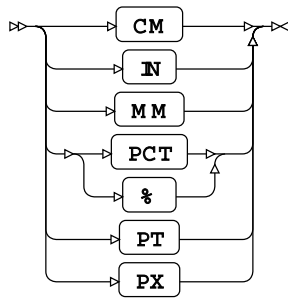
fill-option



line-option



units



FORMAT

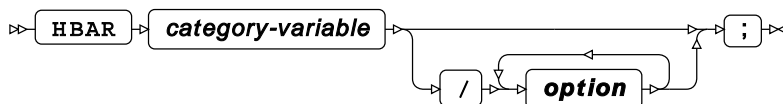
Adds formats to one or more variables in a dataset.



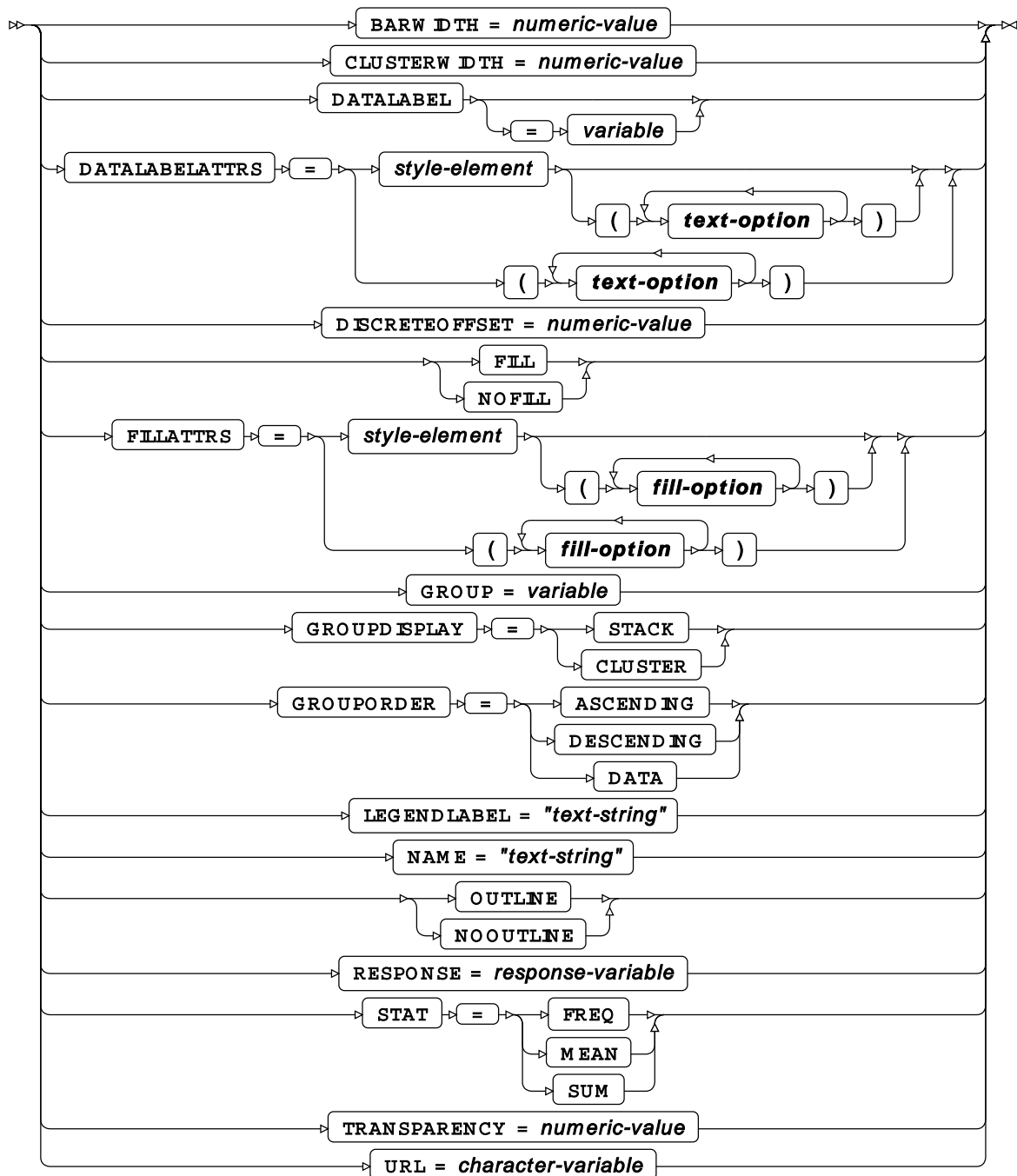
ⁱ See [Variable Lists](#) (page 32).

HBAR

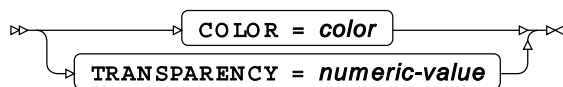
Draws a horizontal bar chart using unsummarised data.



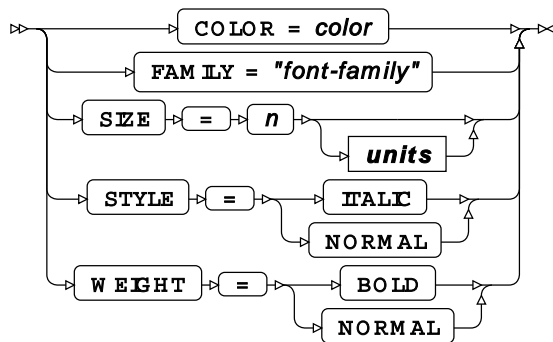
option



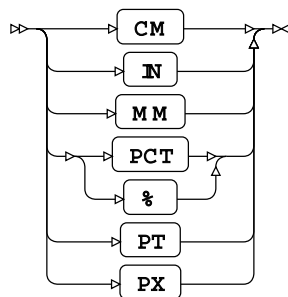
fill-option



text-option



units

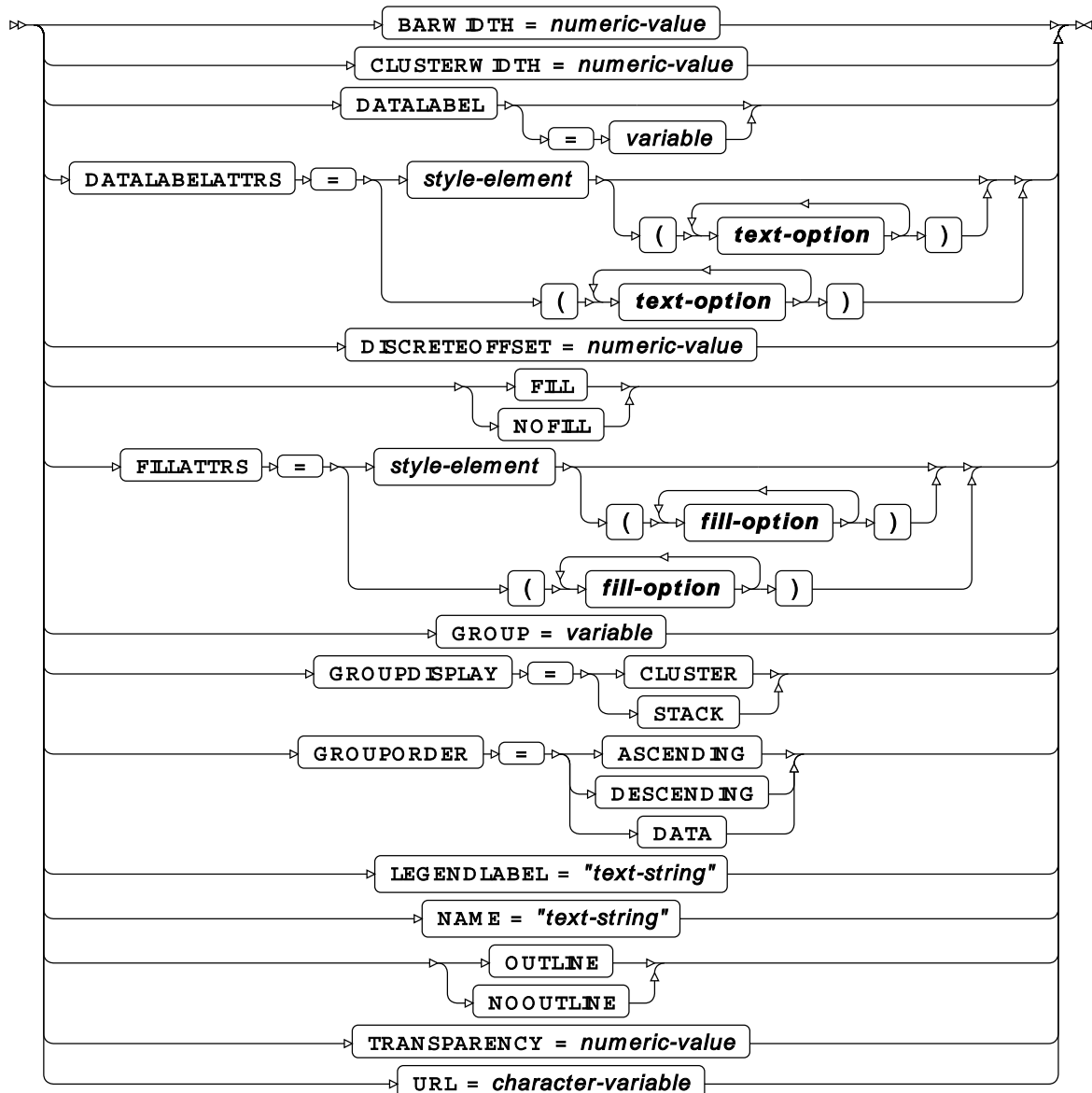


HBARPARAM

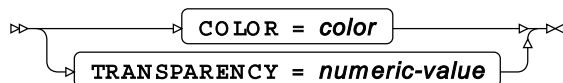
Draws a horizontal bar chart using summarised data.



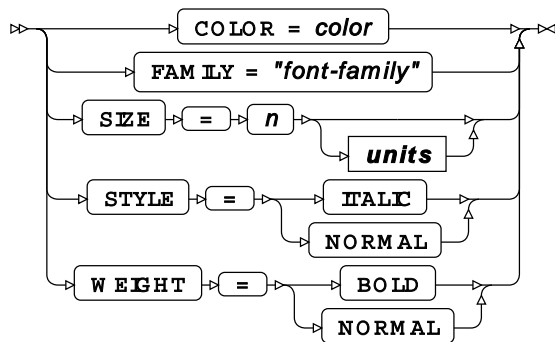
option



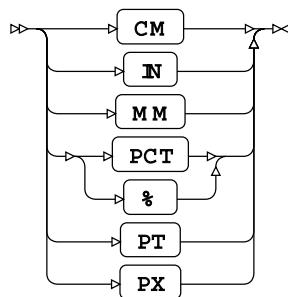
fill-option



text-option

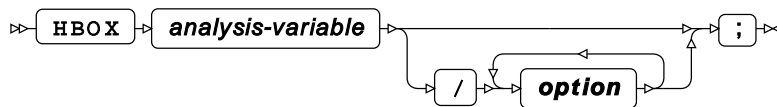


units

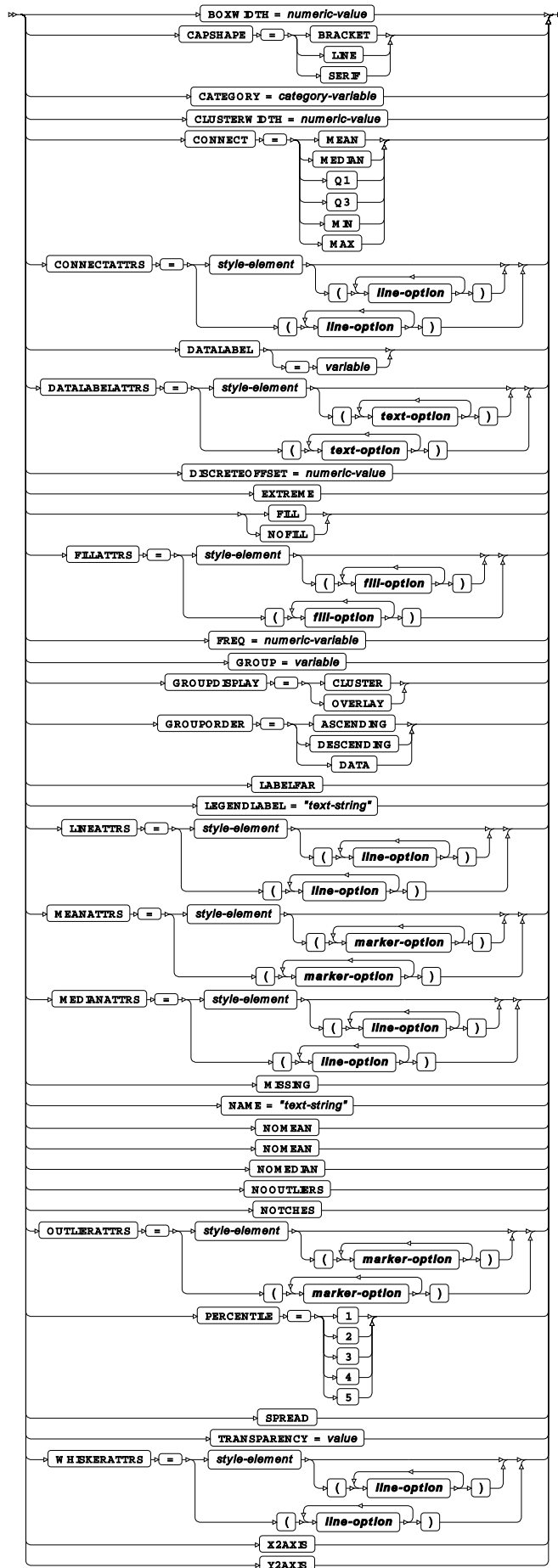


HBOX

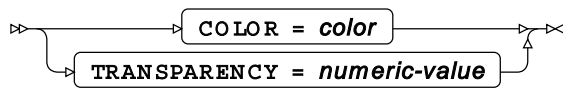
Draws a horizontal box plot.



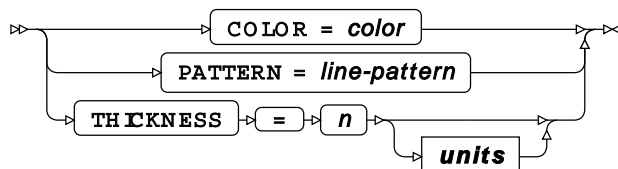
option



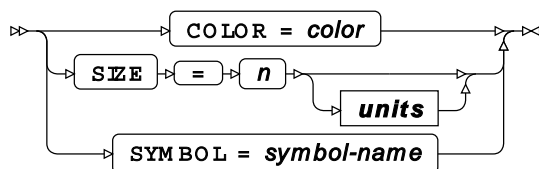
fill-option



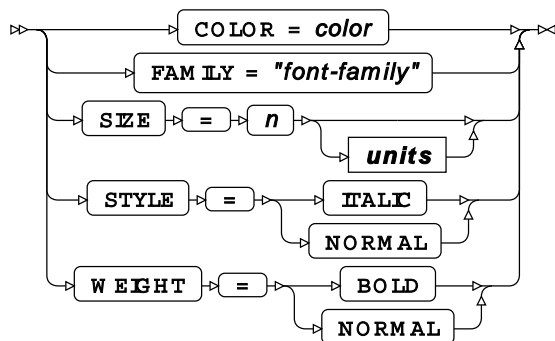
line-option



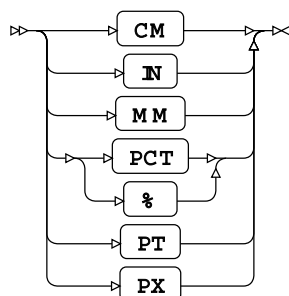
marker-option



text-option

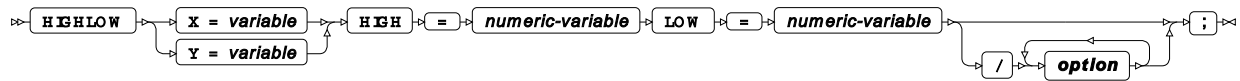


units

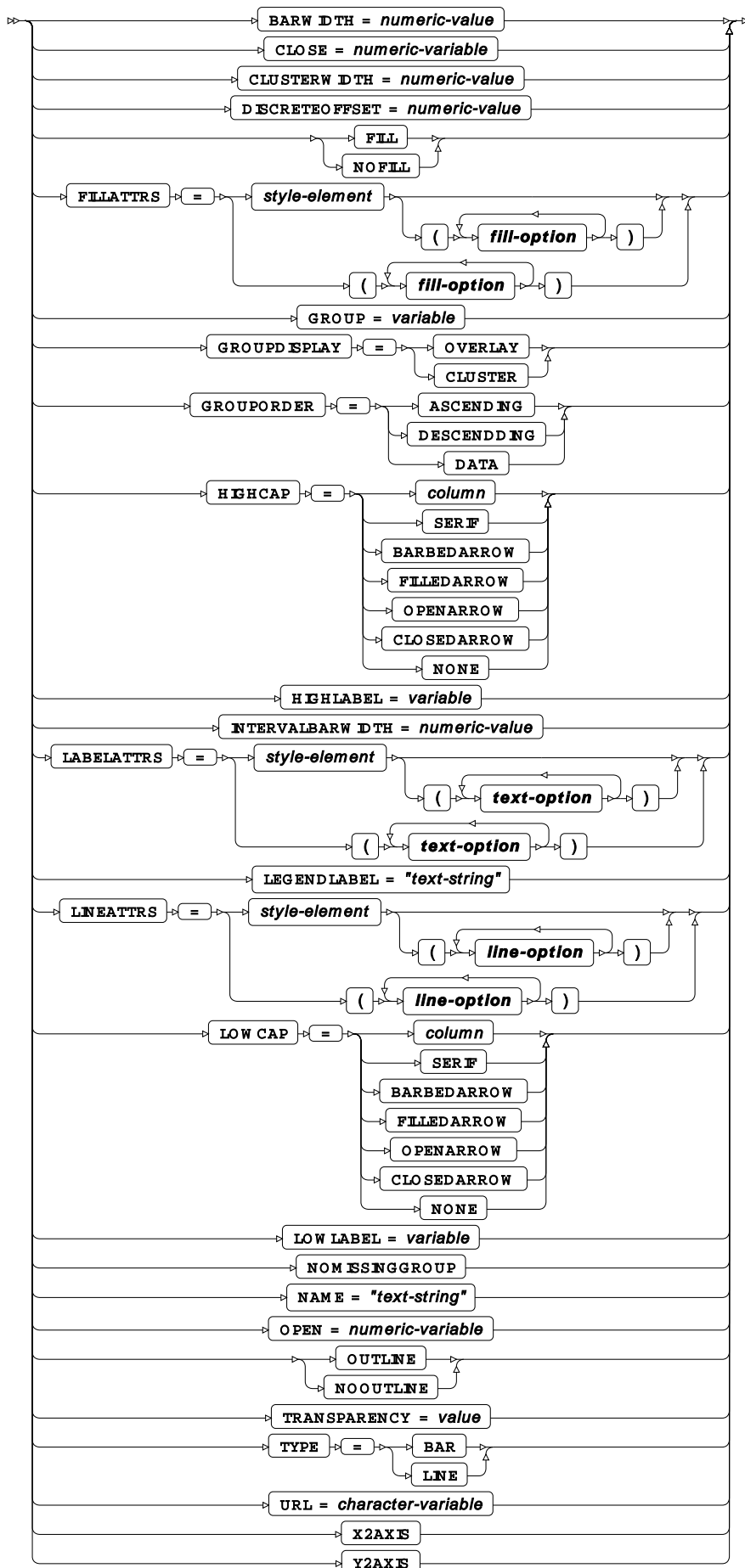


HIGHLOW

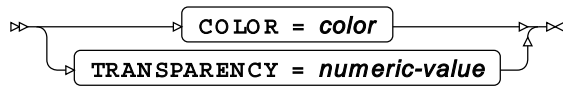
Draws a high-low plot for categorised data.



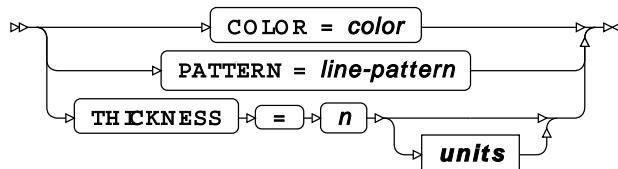
option



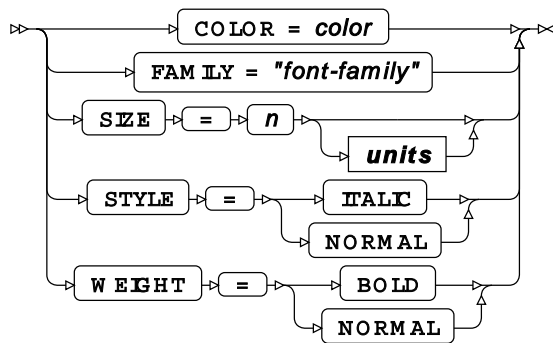
fill-option



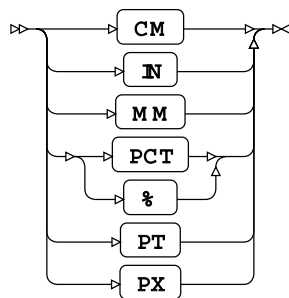
line-option



text-option

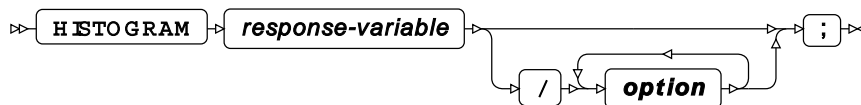


units

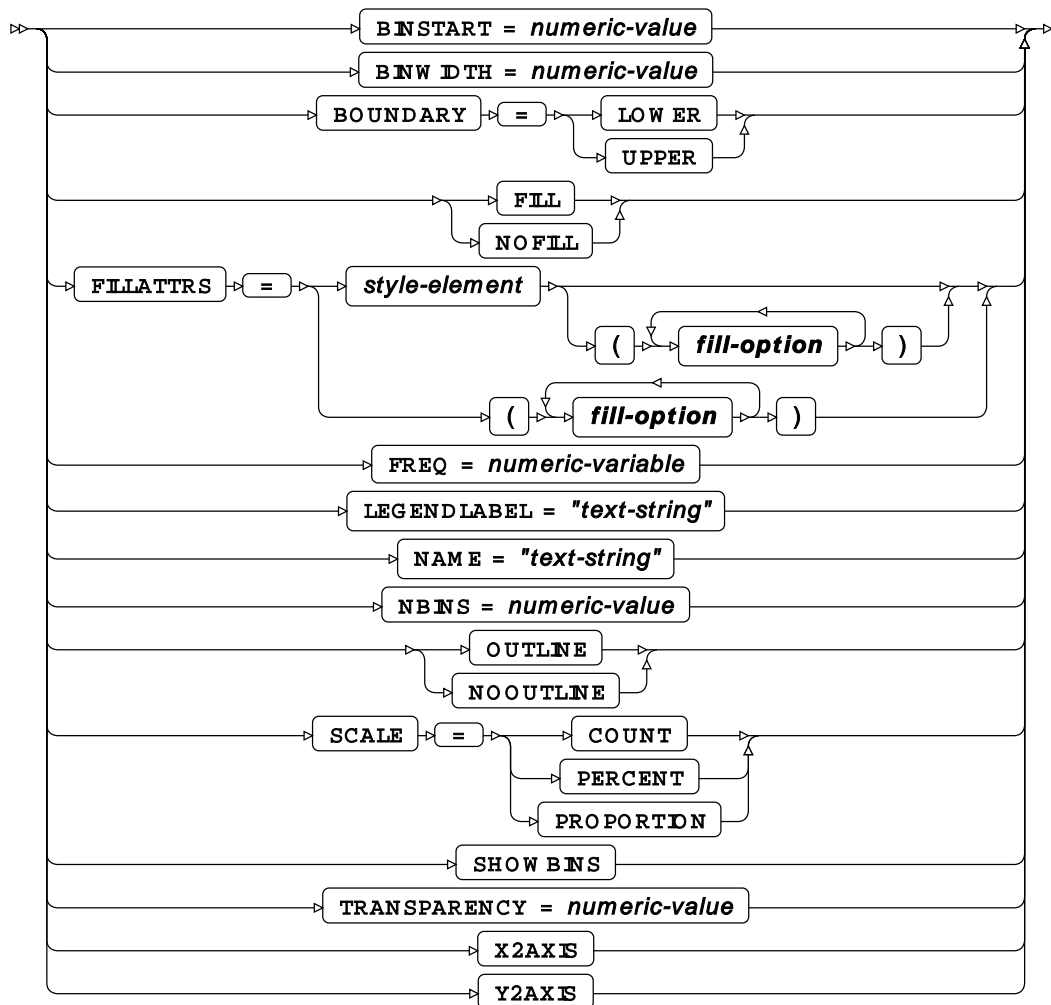


HISTOGRAM

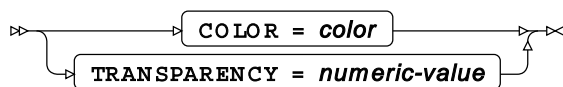
Draws a histogram



option

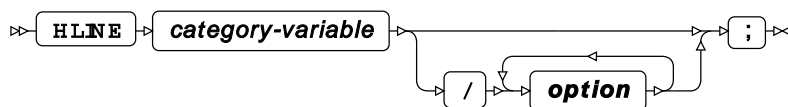


fill-option

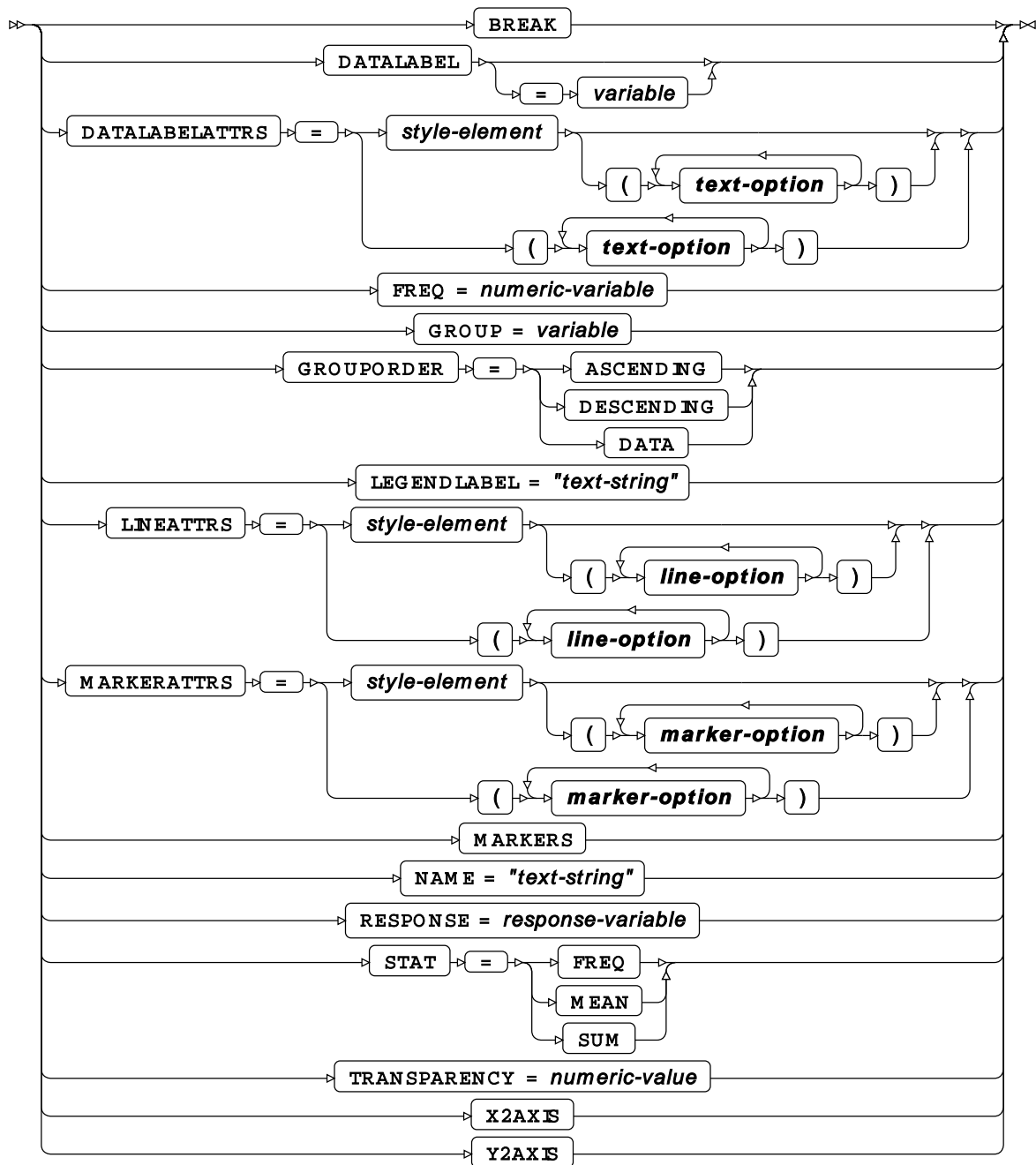


HLINE

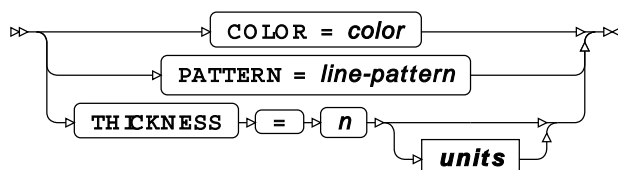
Draws a horizontal line plot using unsummarised data.



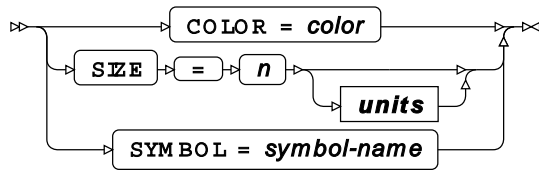
option



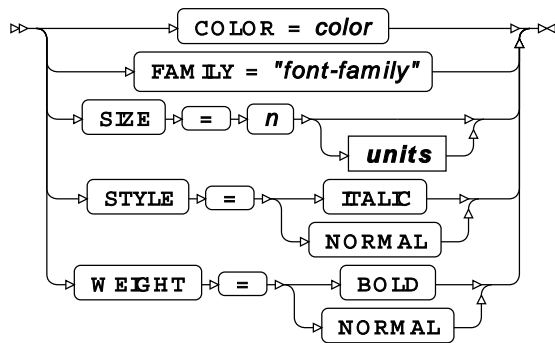
line-option



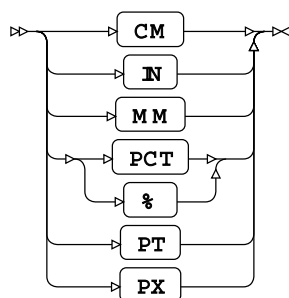
marker-option



text-option

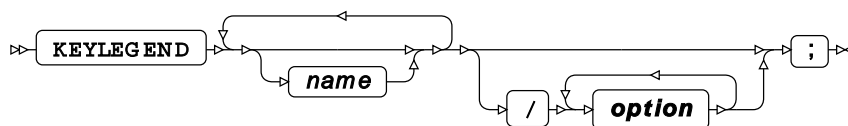


units

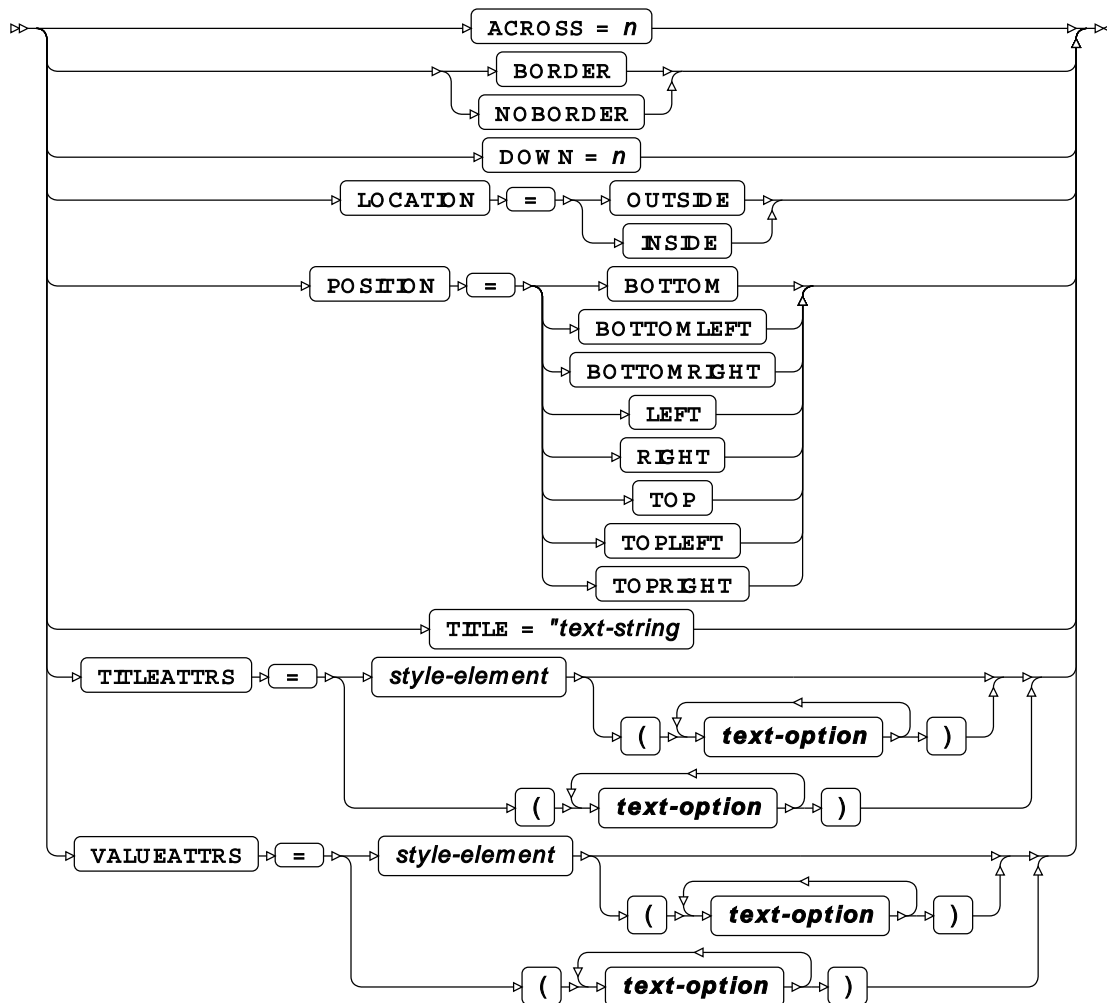


KEYLEGEND

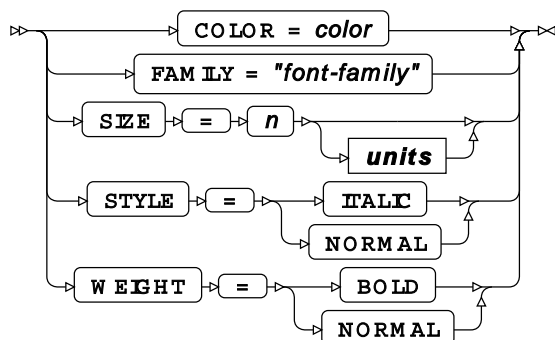
Specifies a legend and adds it to a plot.



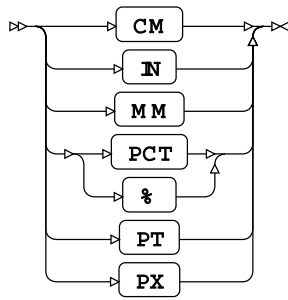
option



text-option

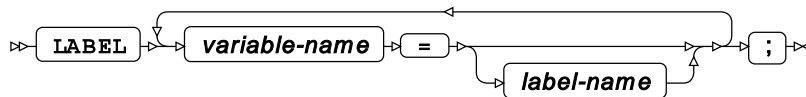


units



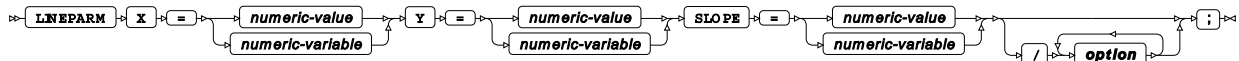
LABEL

Adds labels to one or more variables in a dataset.

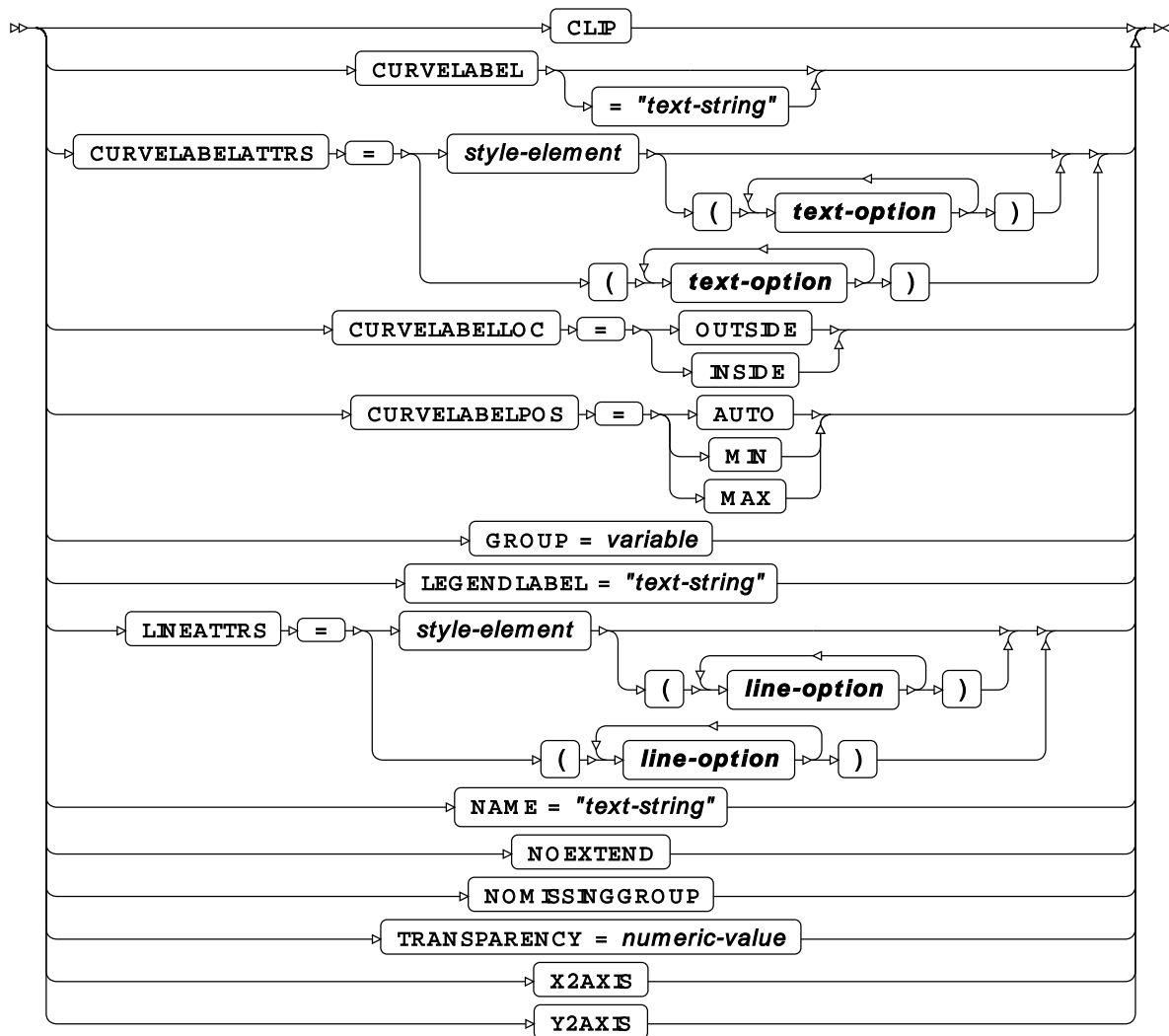


LINEPARM

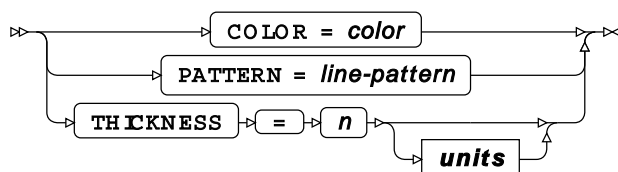
Draws one or more straight lines each defined by a point and gradient.



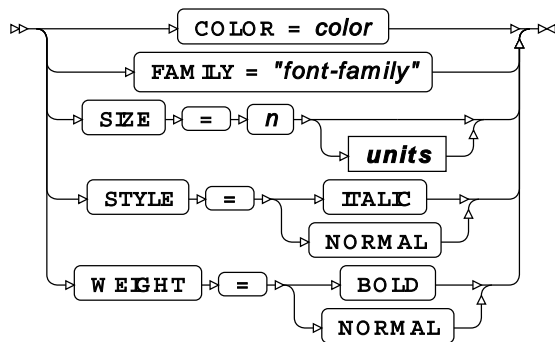
option



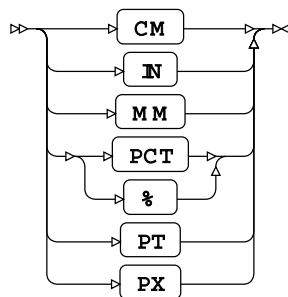
line-option



text-option

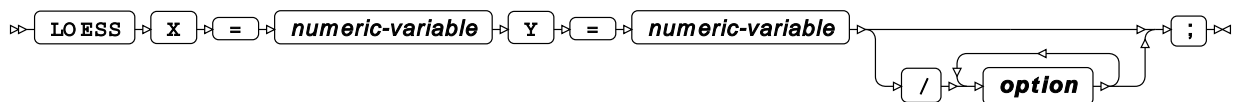


units

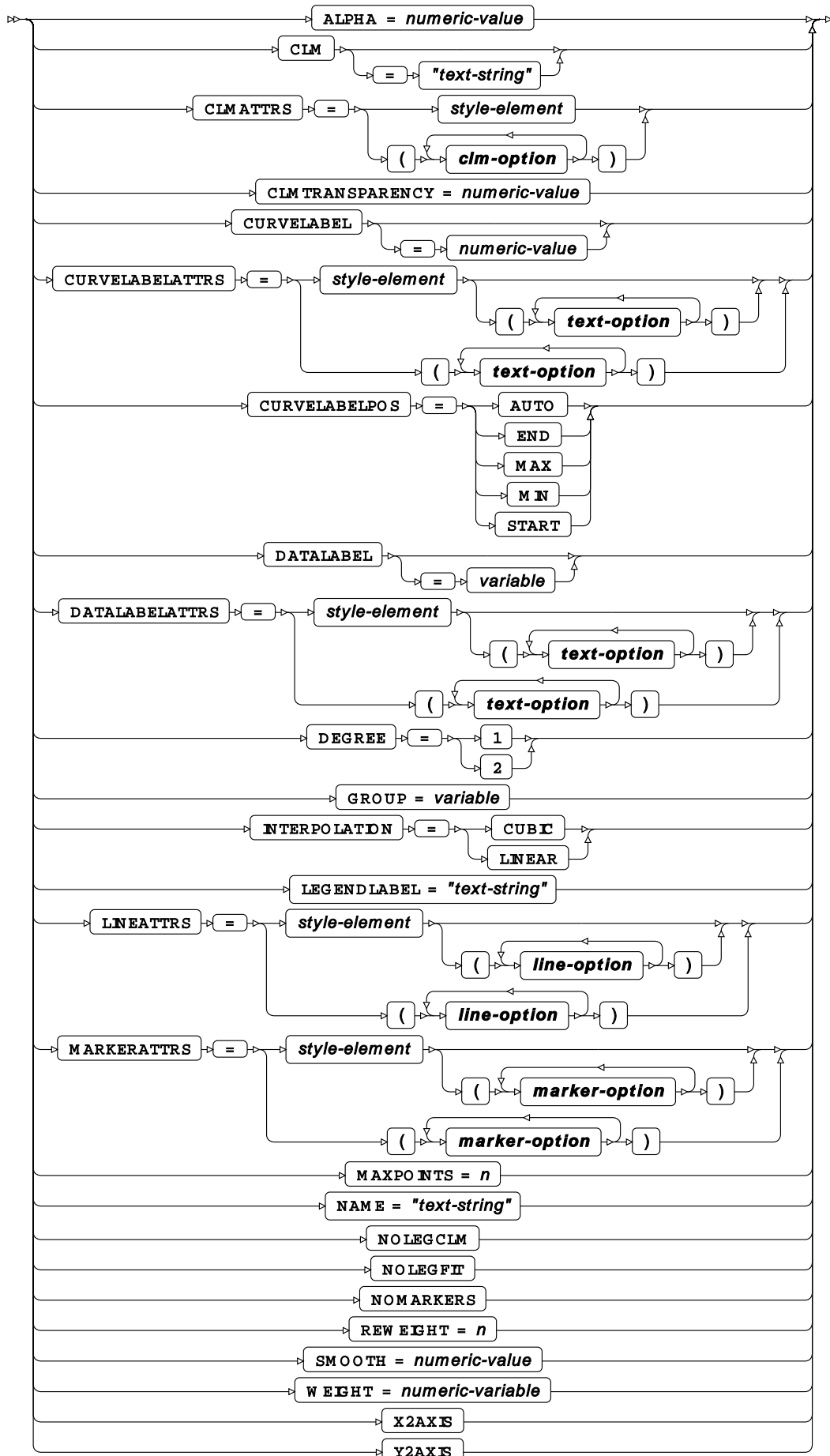


LOESS

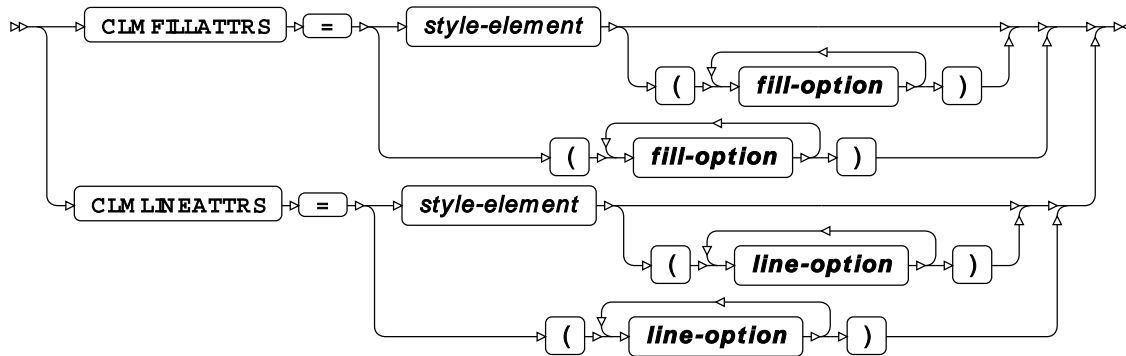
Draws a fitted loess curve.



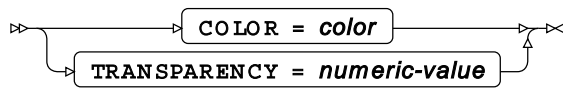
option



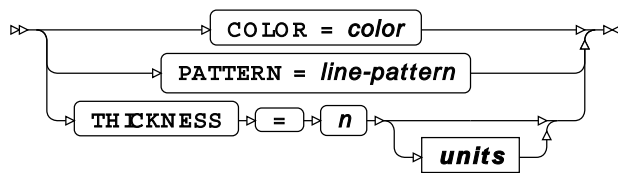
clm-option



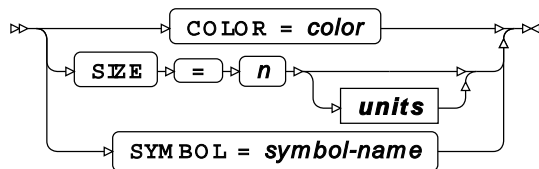
fill-option



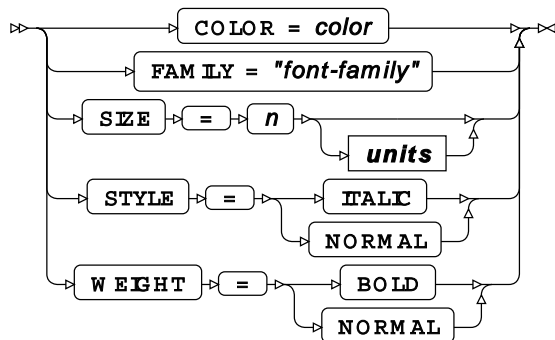
line-option



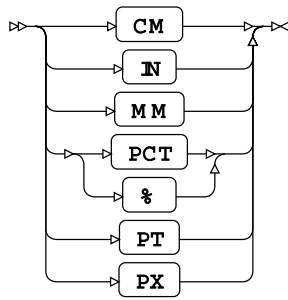
marker-option



text-option

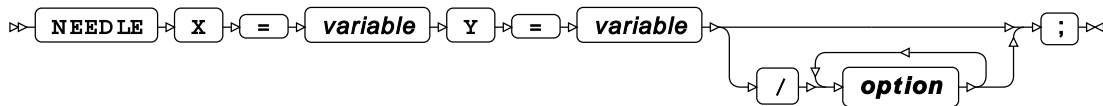


units

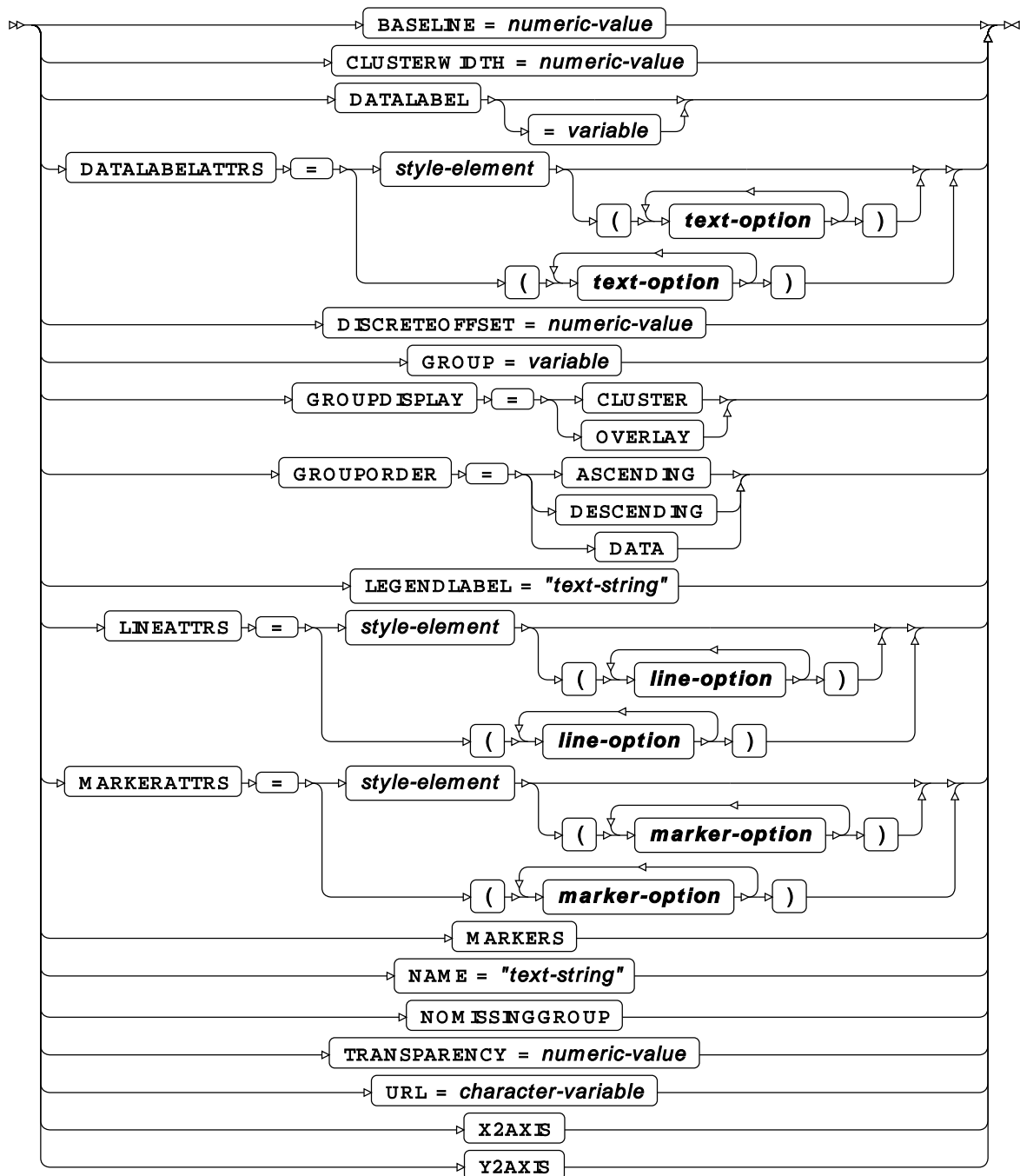


NEEDLE

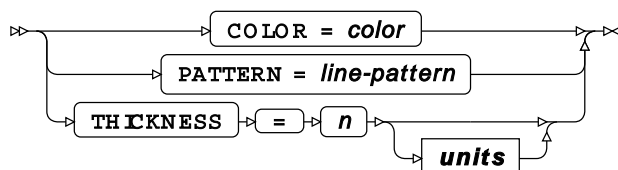
Draws a needle plot.



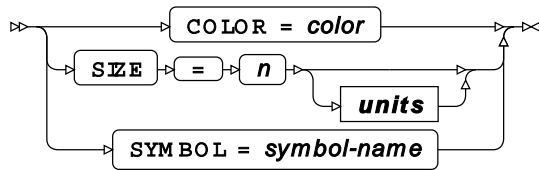
option



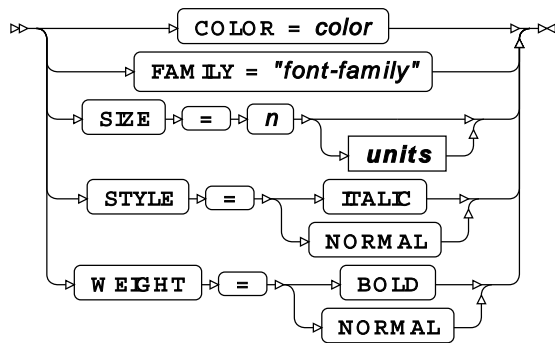
line-option



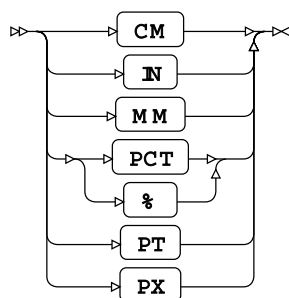
marker-option



text-option

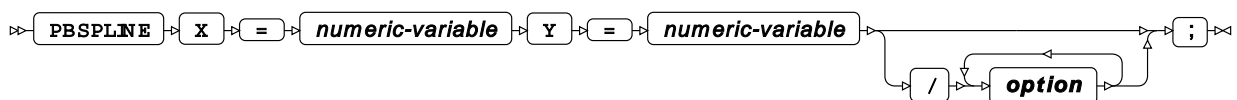


units

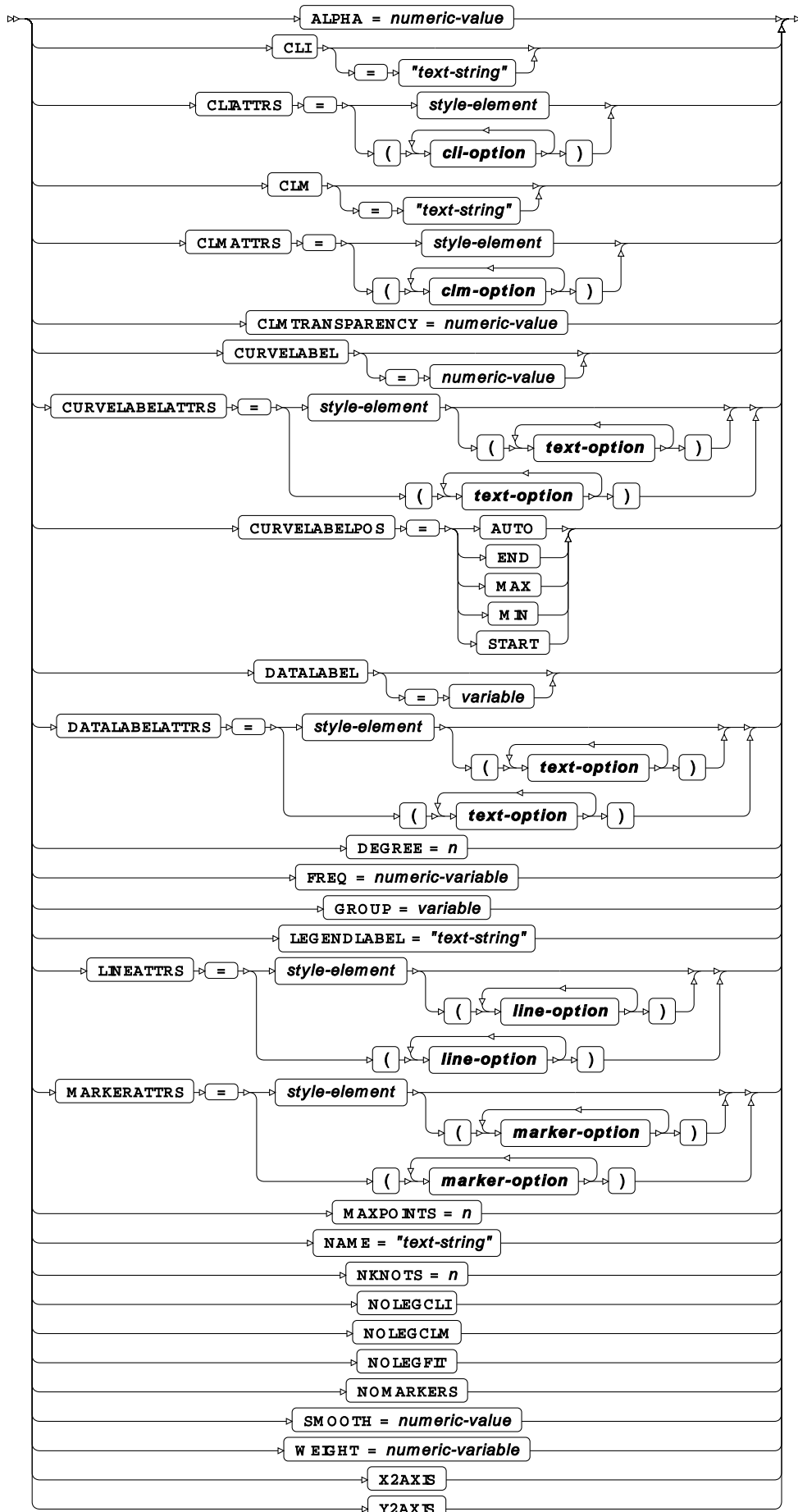


PBSPLINE

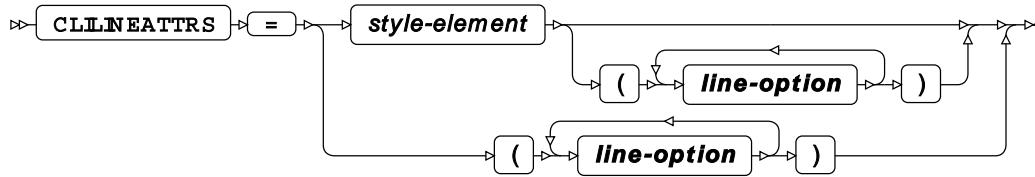
Draws a fitted, penalised B-spline curve.



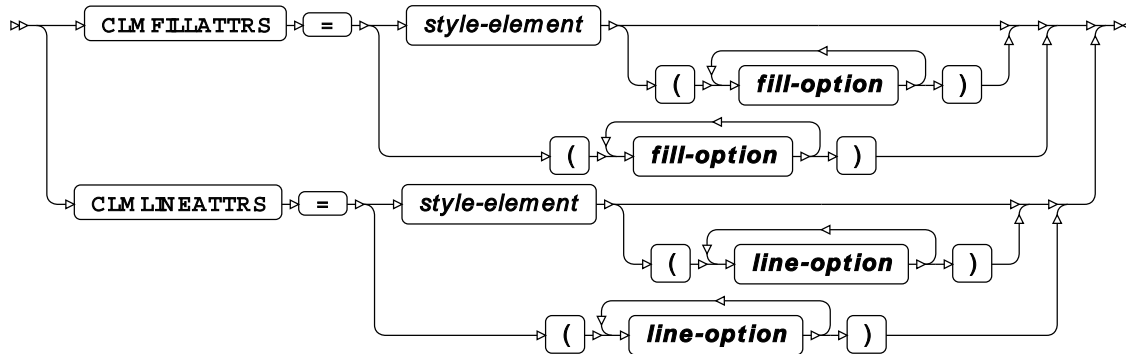
option



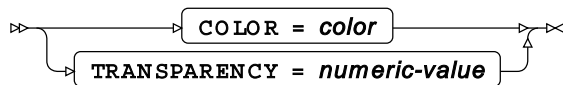
cli-option



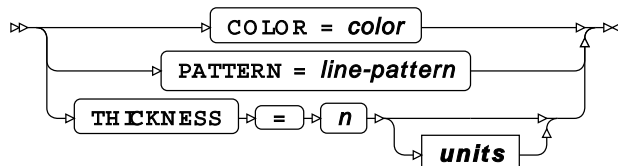
clm-option



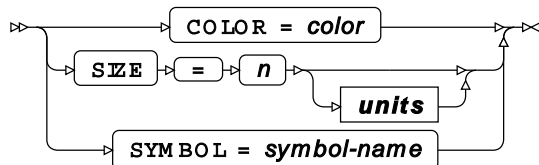
fill-option



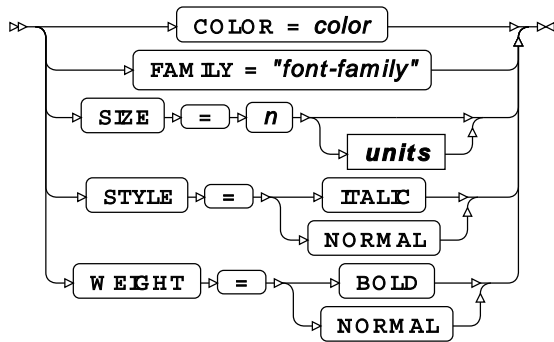
line-option



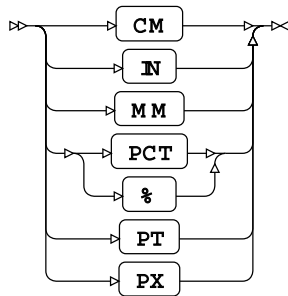
marker-option



text-option

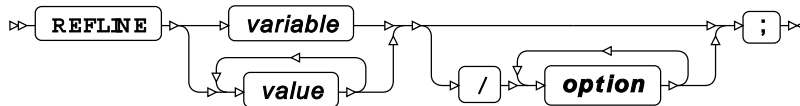


units

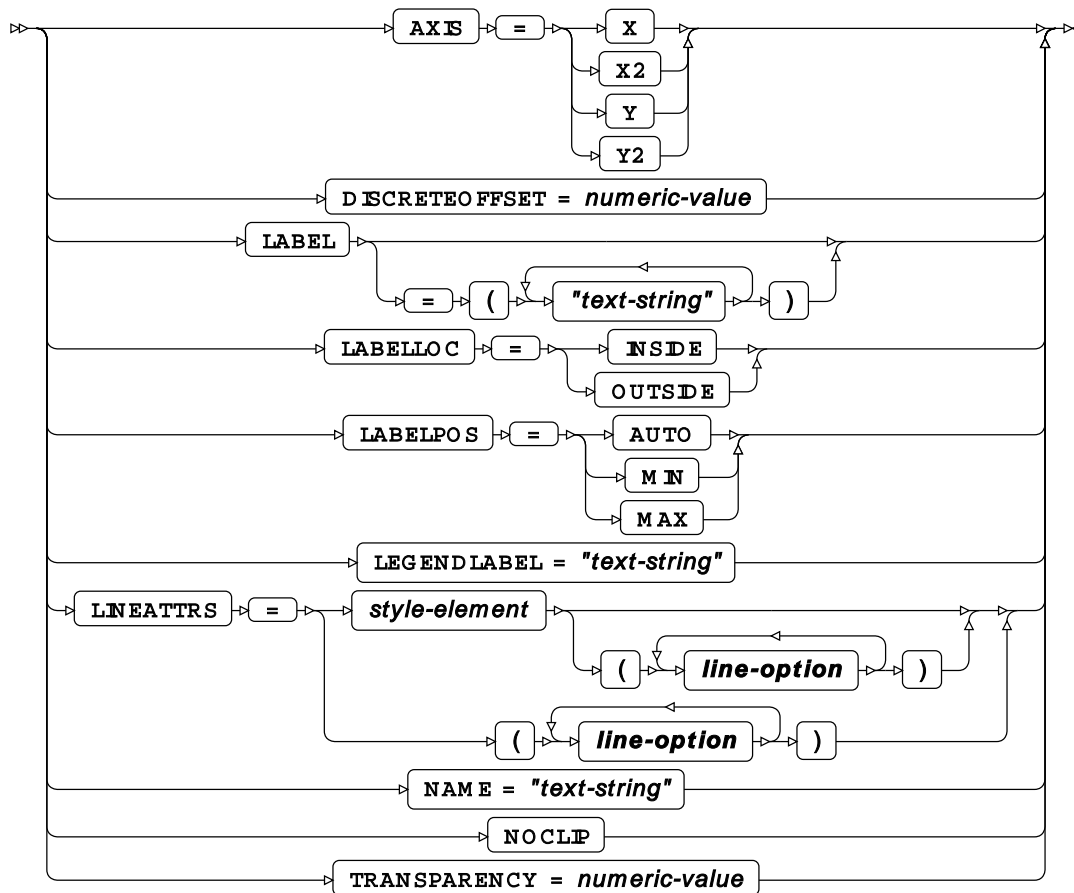


REFLINE

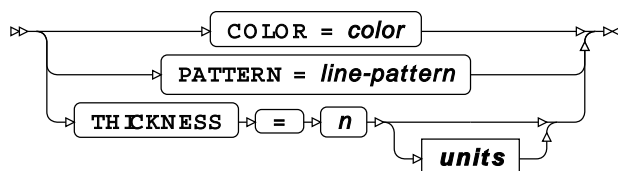
Draws one or more horizontal or vertical reference lines.



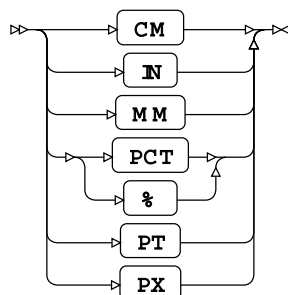
option



line-option

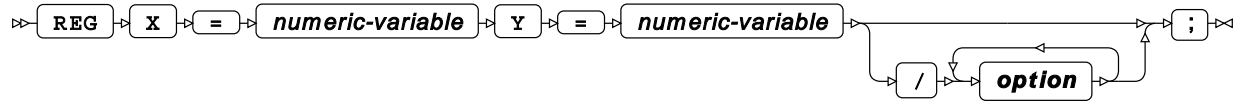


units

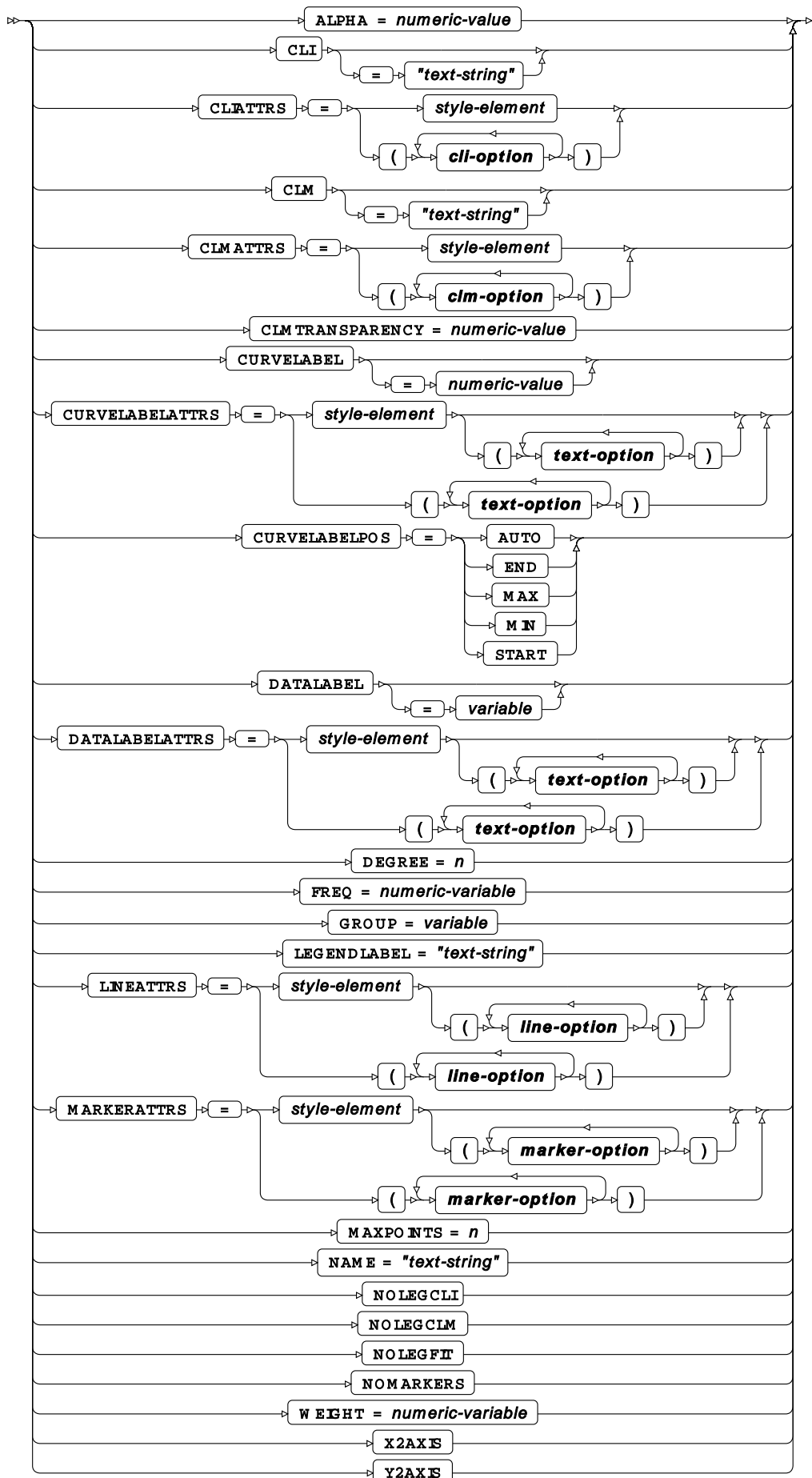


REG

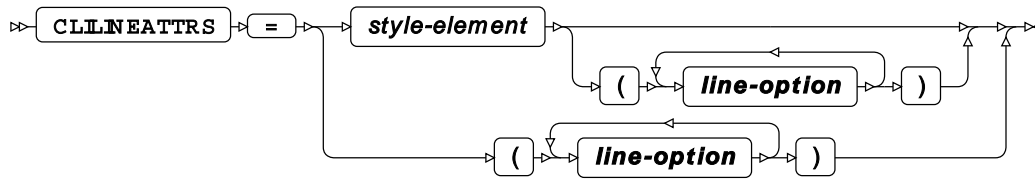
Draws a fitted regression line or curve.



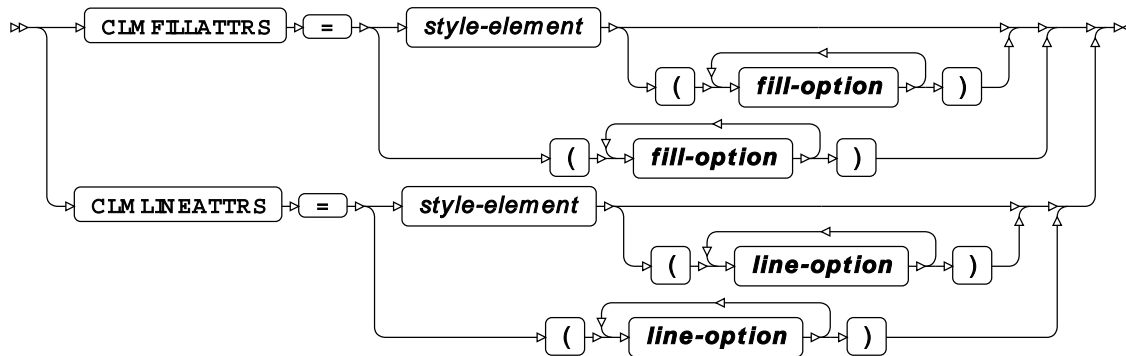
option



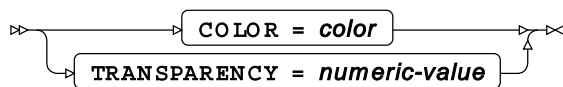
cli-option



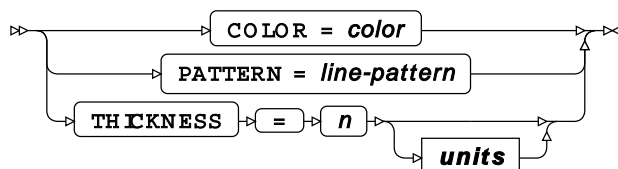
clm-option



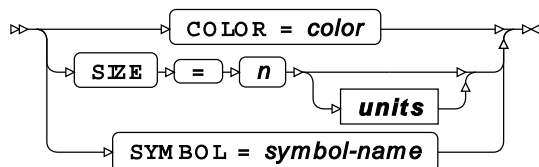
fill-option



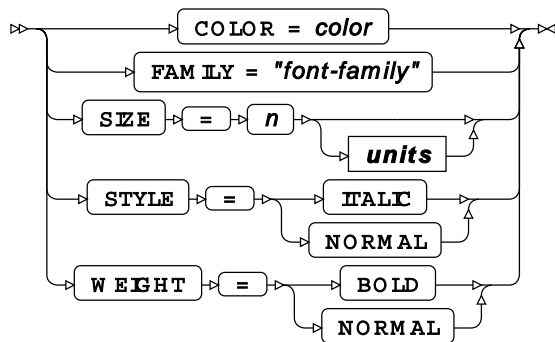
line-option



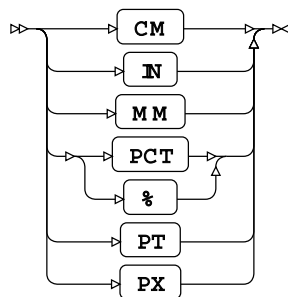
marker-option



text-option

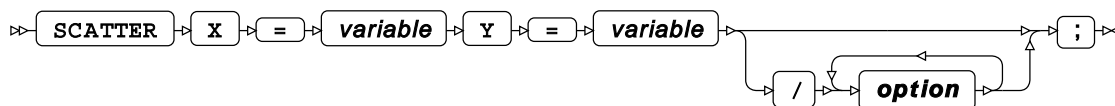


units

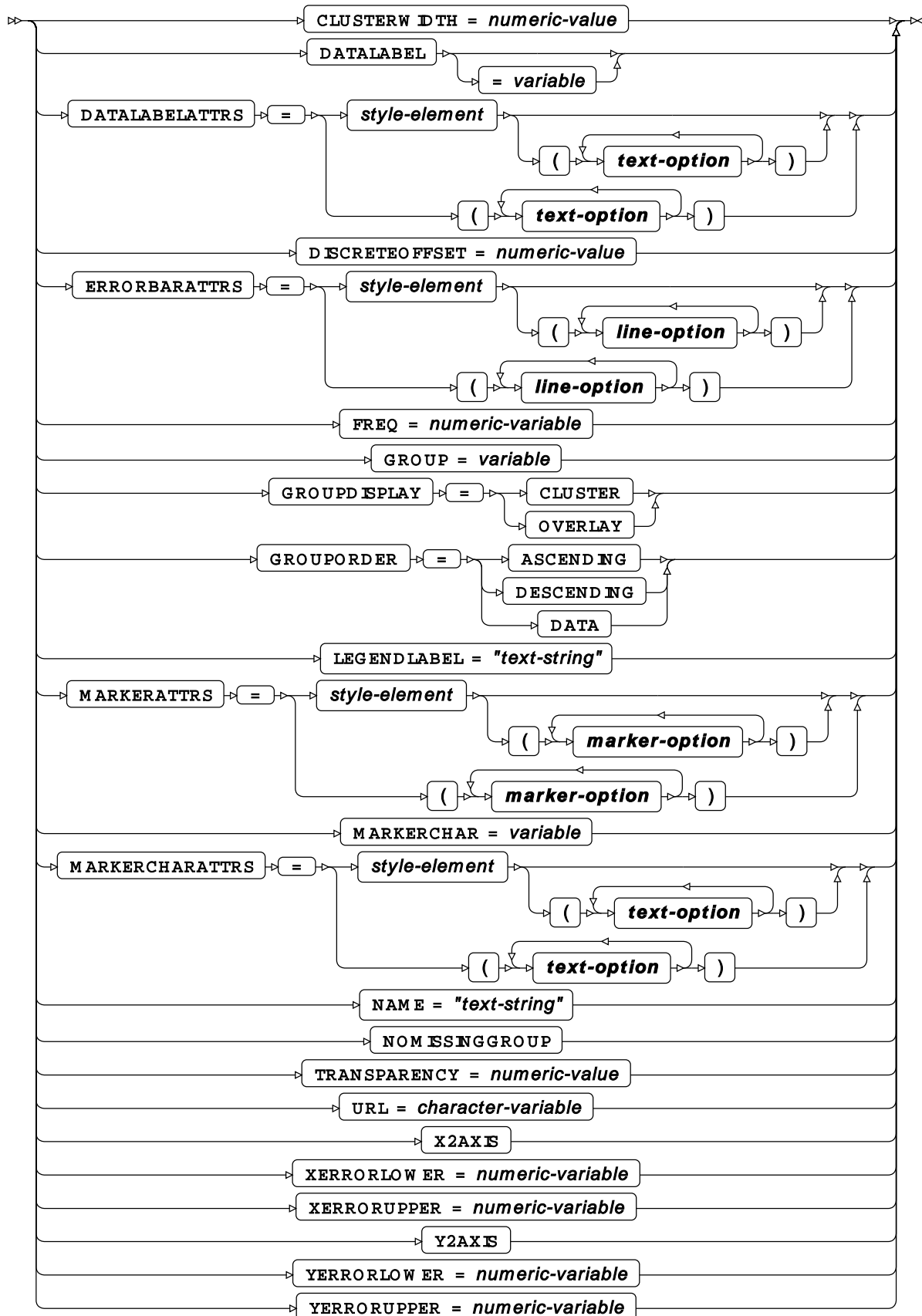


SCATTER

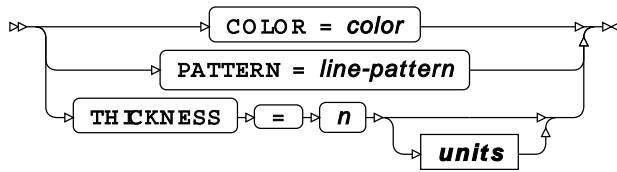
Draws a scatter plot.



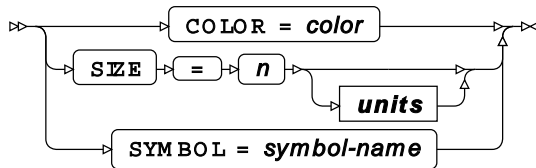
option



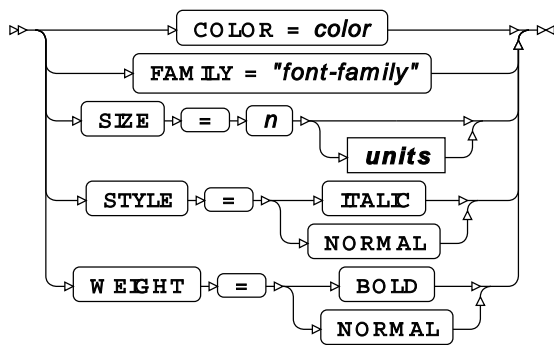
line-option



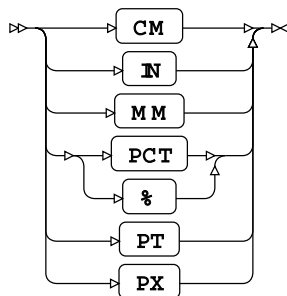
marker-option



text-option

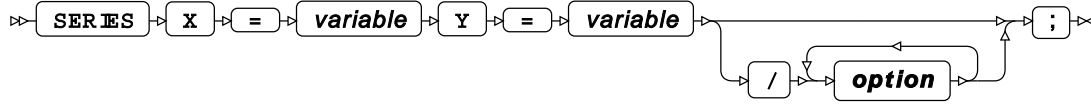


units

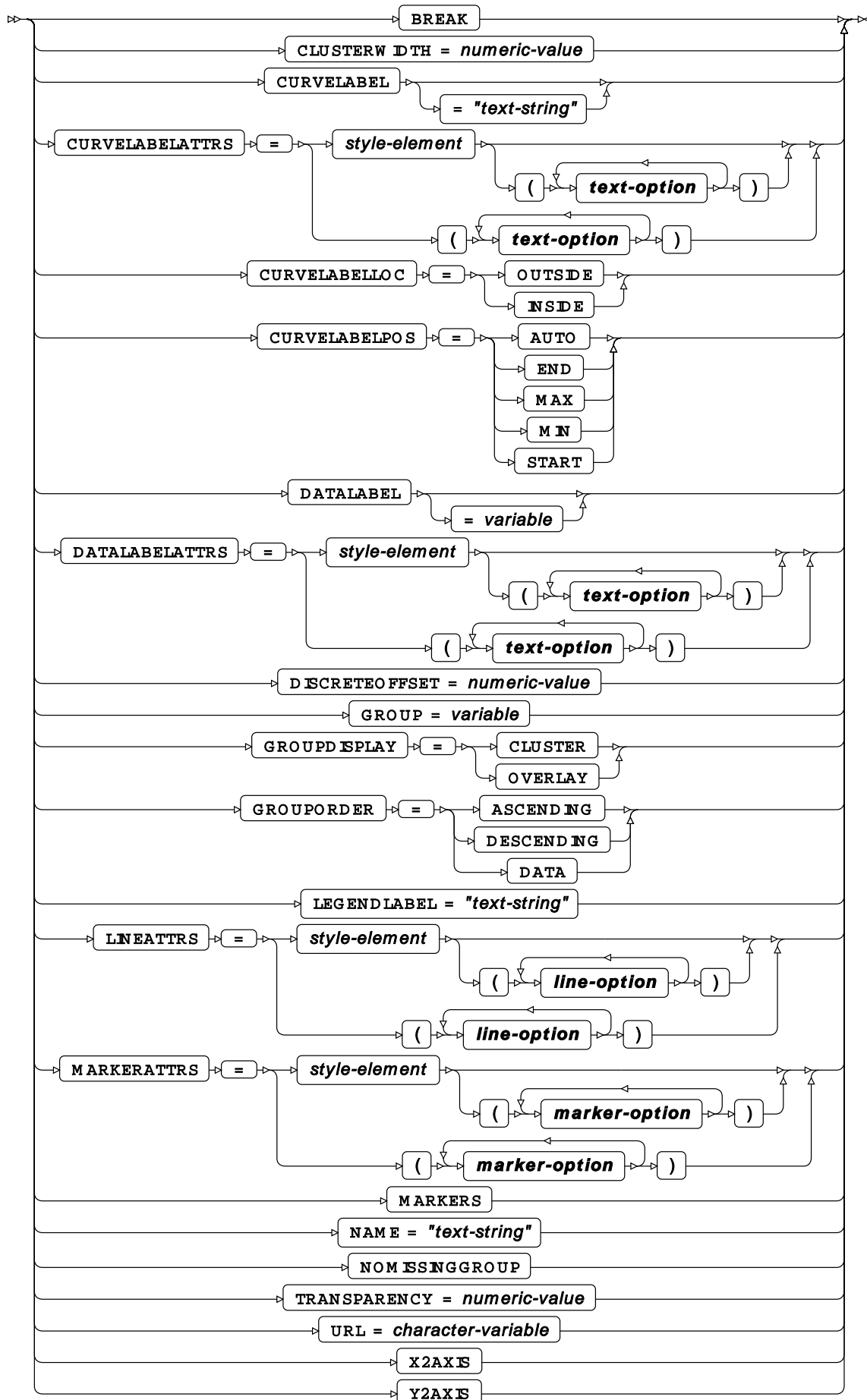


SERIES

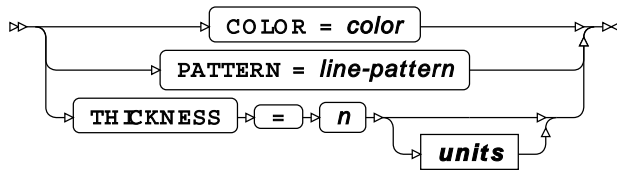
Draws a series plot using unsummarised data.



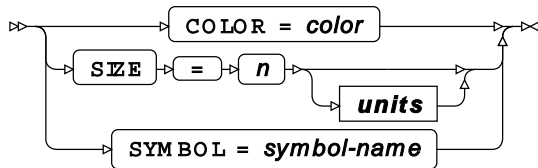
option



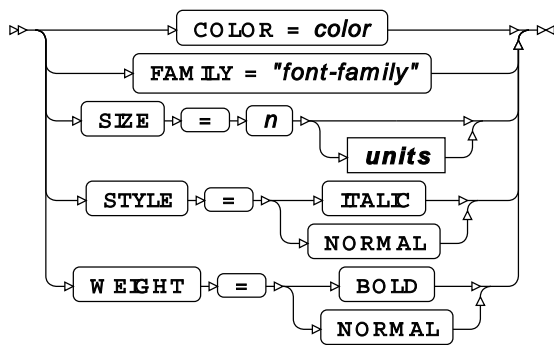
line-option



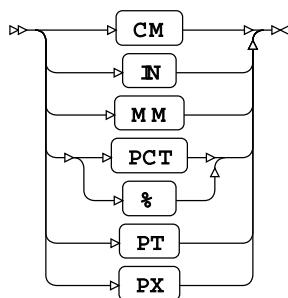
marker-option



text-option

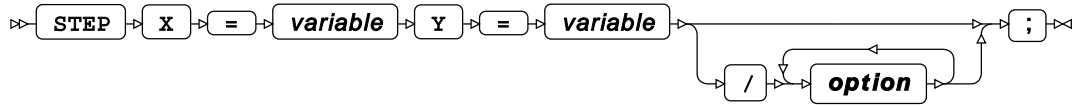


units

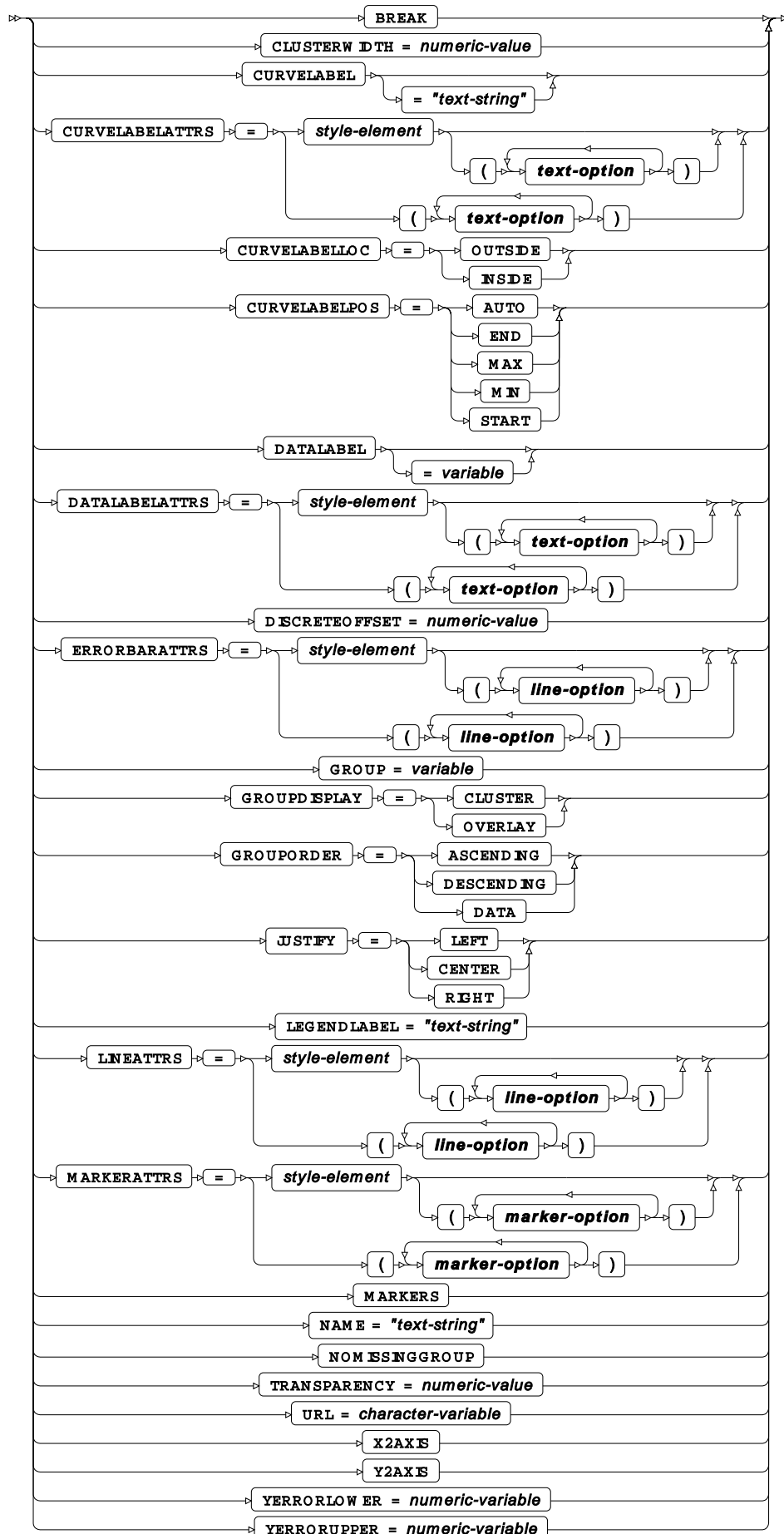


STEP

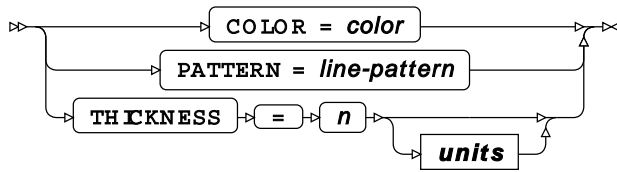
Draws a step plot.



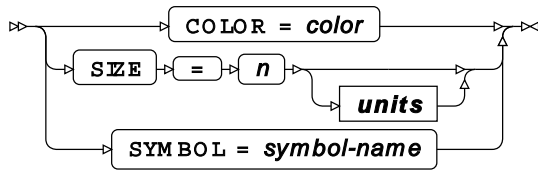
option



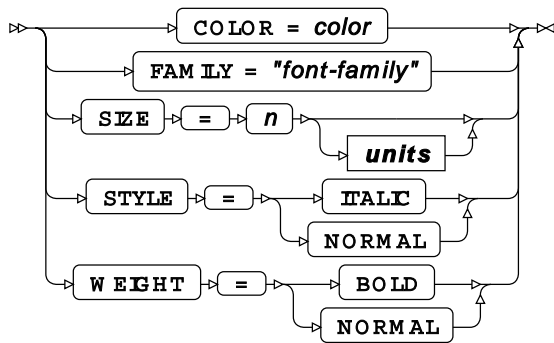
line-option



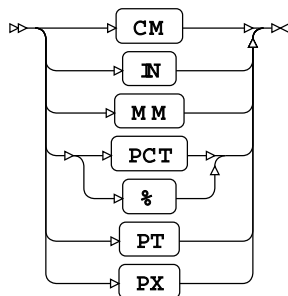
marker-option



text-option

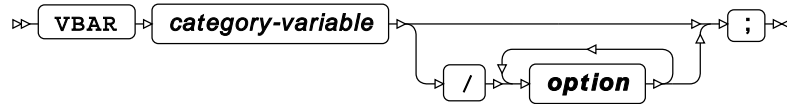


units

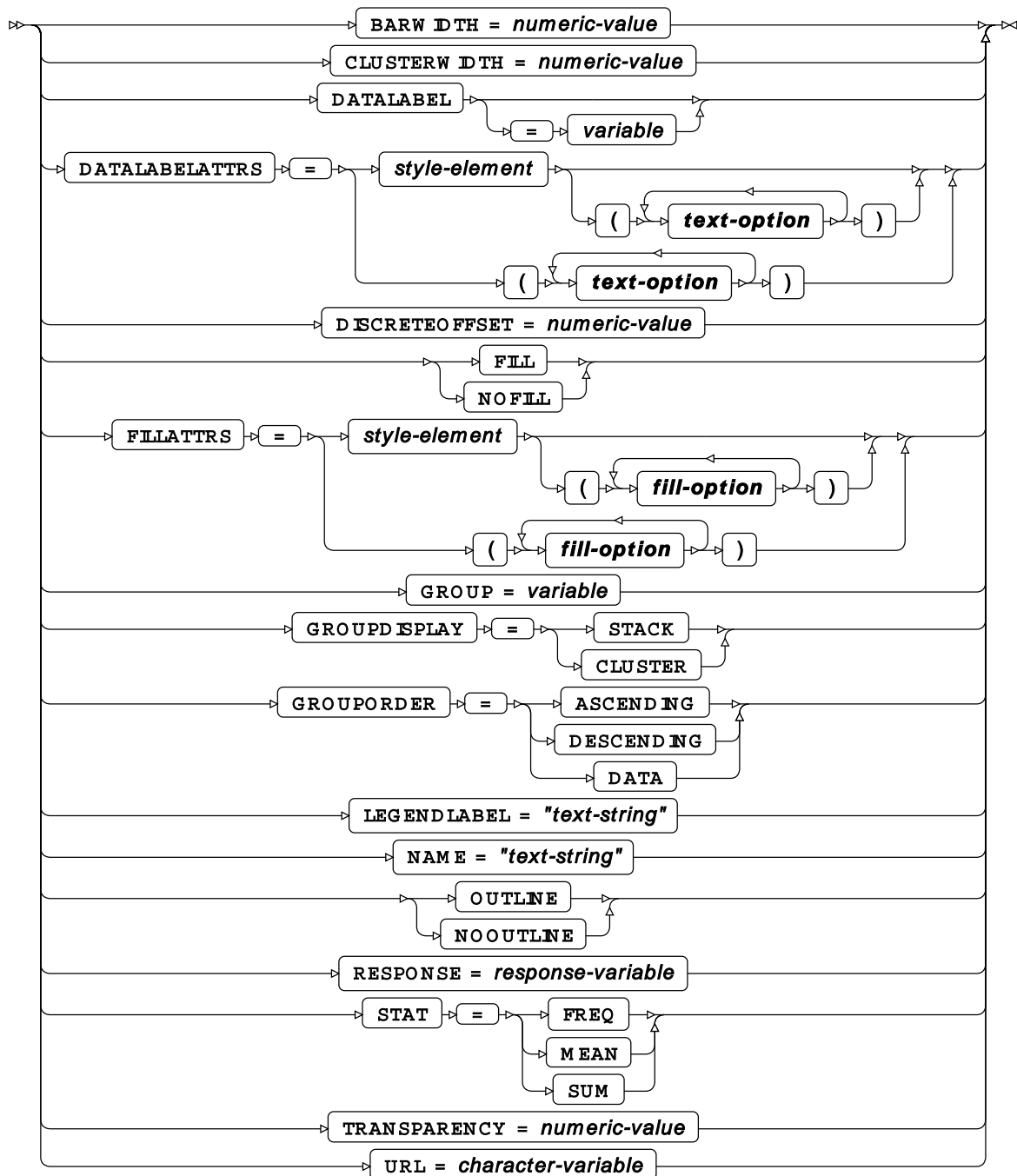


VBAR

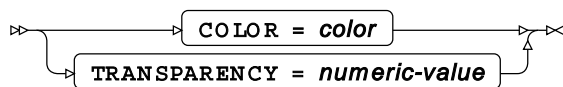
Draws a vertical bar chart using unsummarised data.



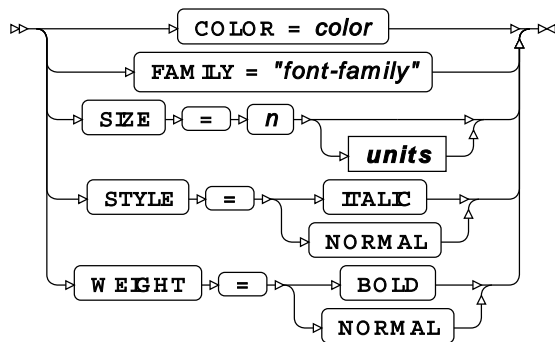
option



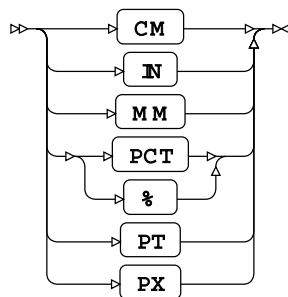
fill-option



text-option

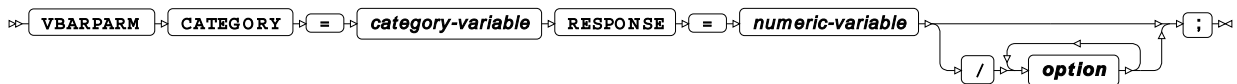


units

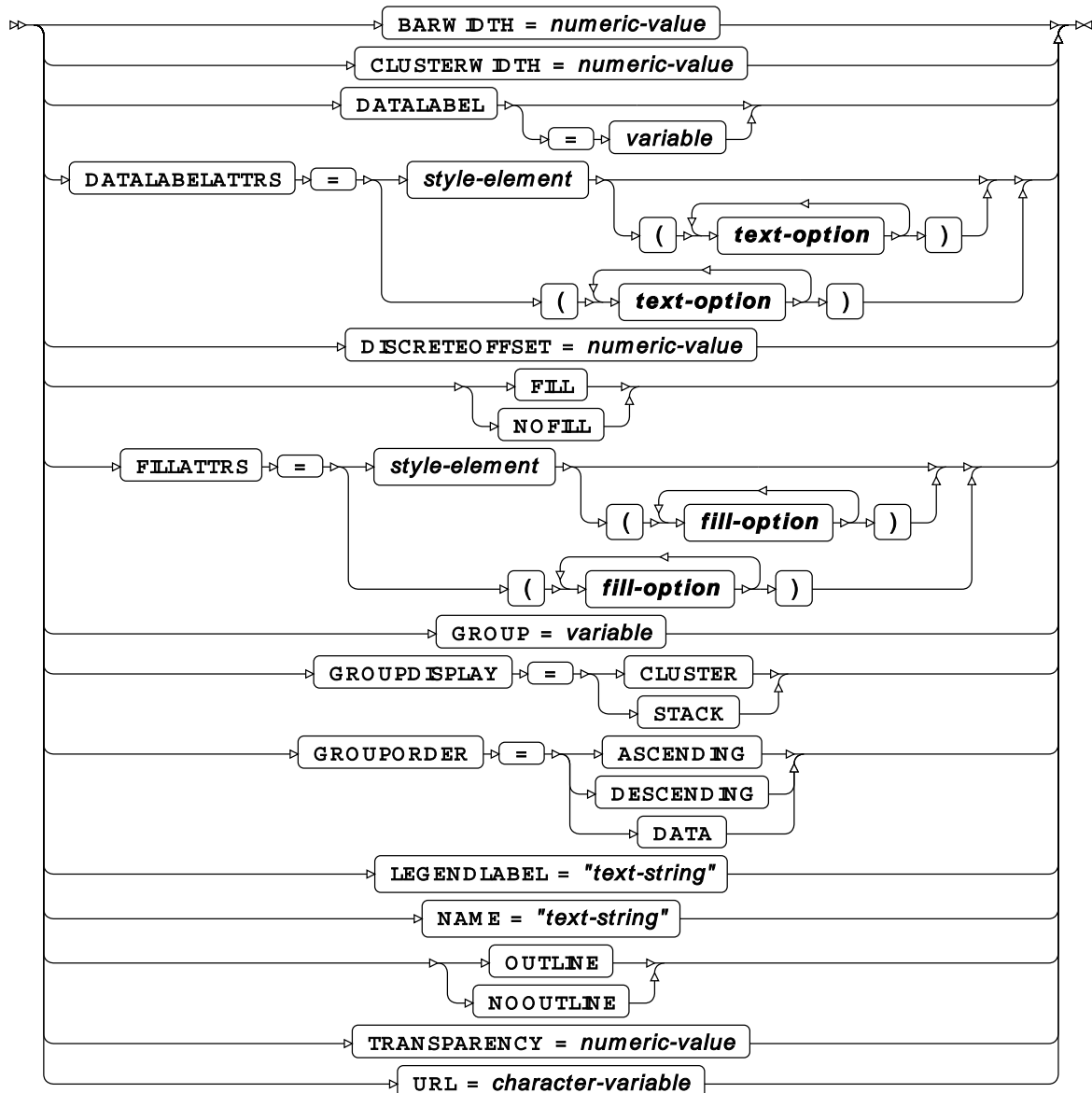


VBARPARG

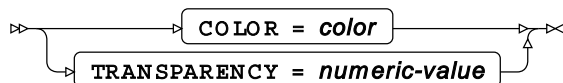
Draws a vertical bar chart using summarised data.



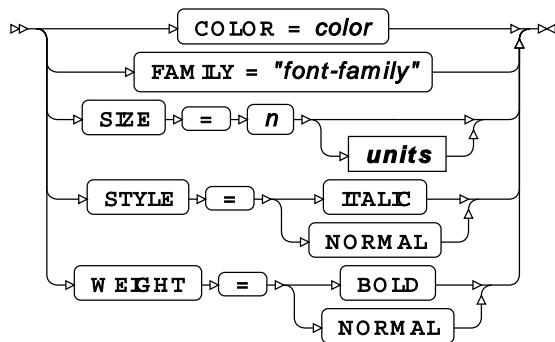
option



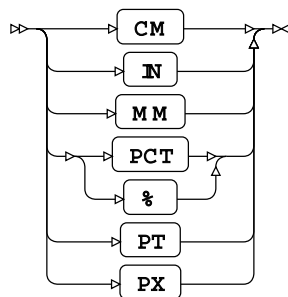
fill-option



text-option

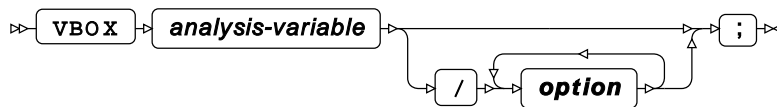


units

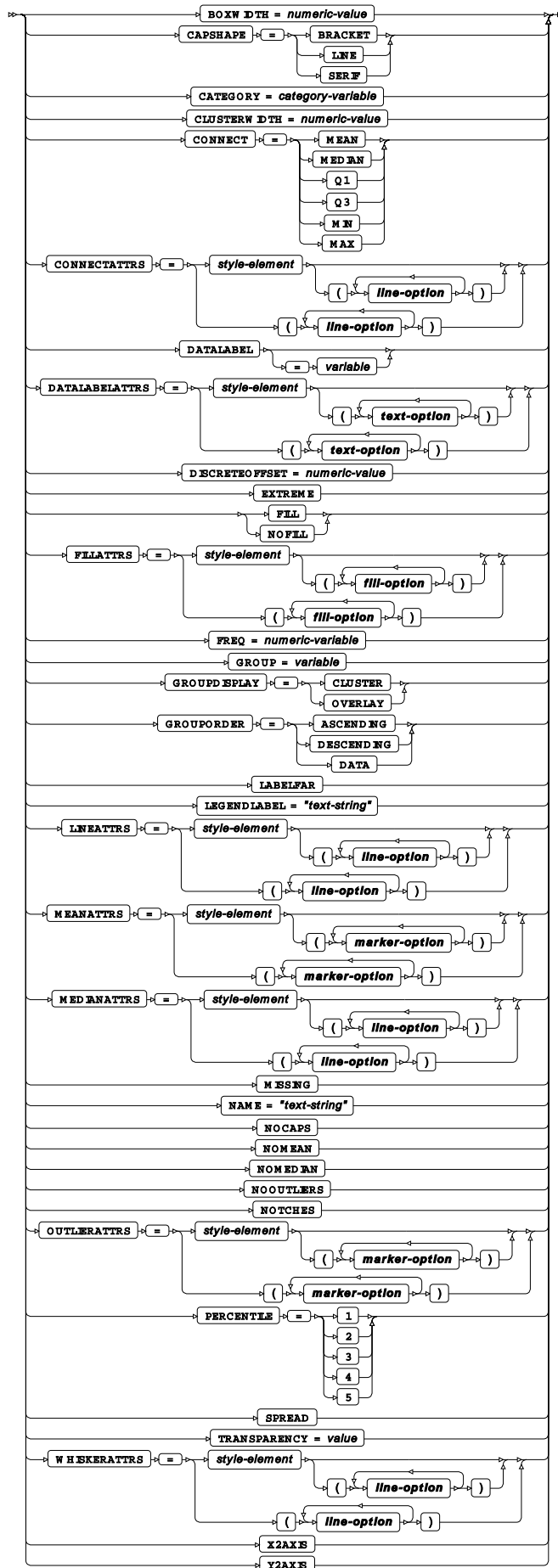


VBOX

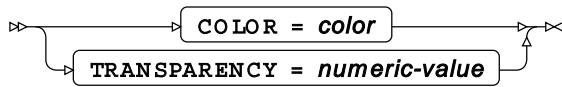
Draws a vertical box plot.



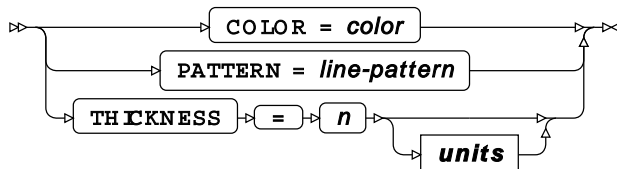
option



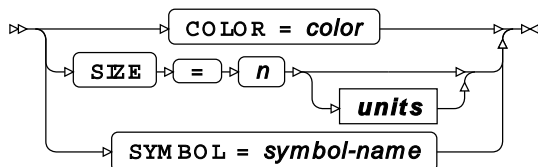
fill-option



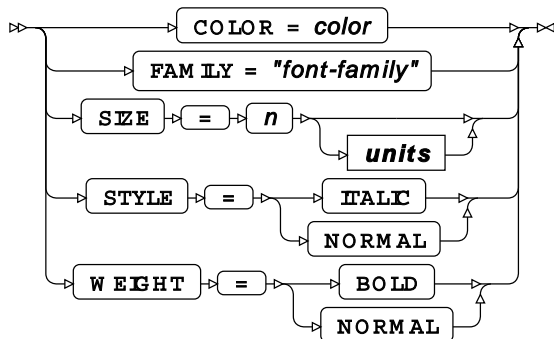
line-option



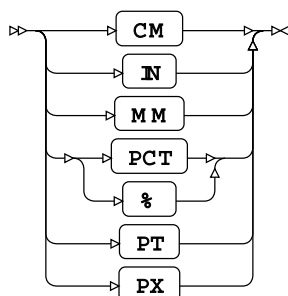
marker-option



text-option

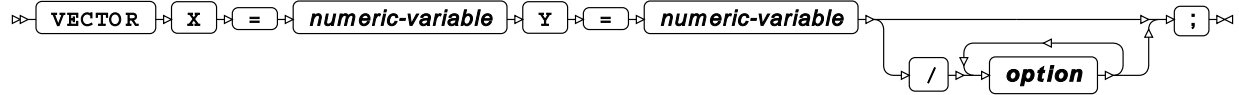


units

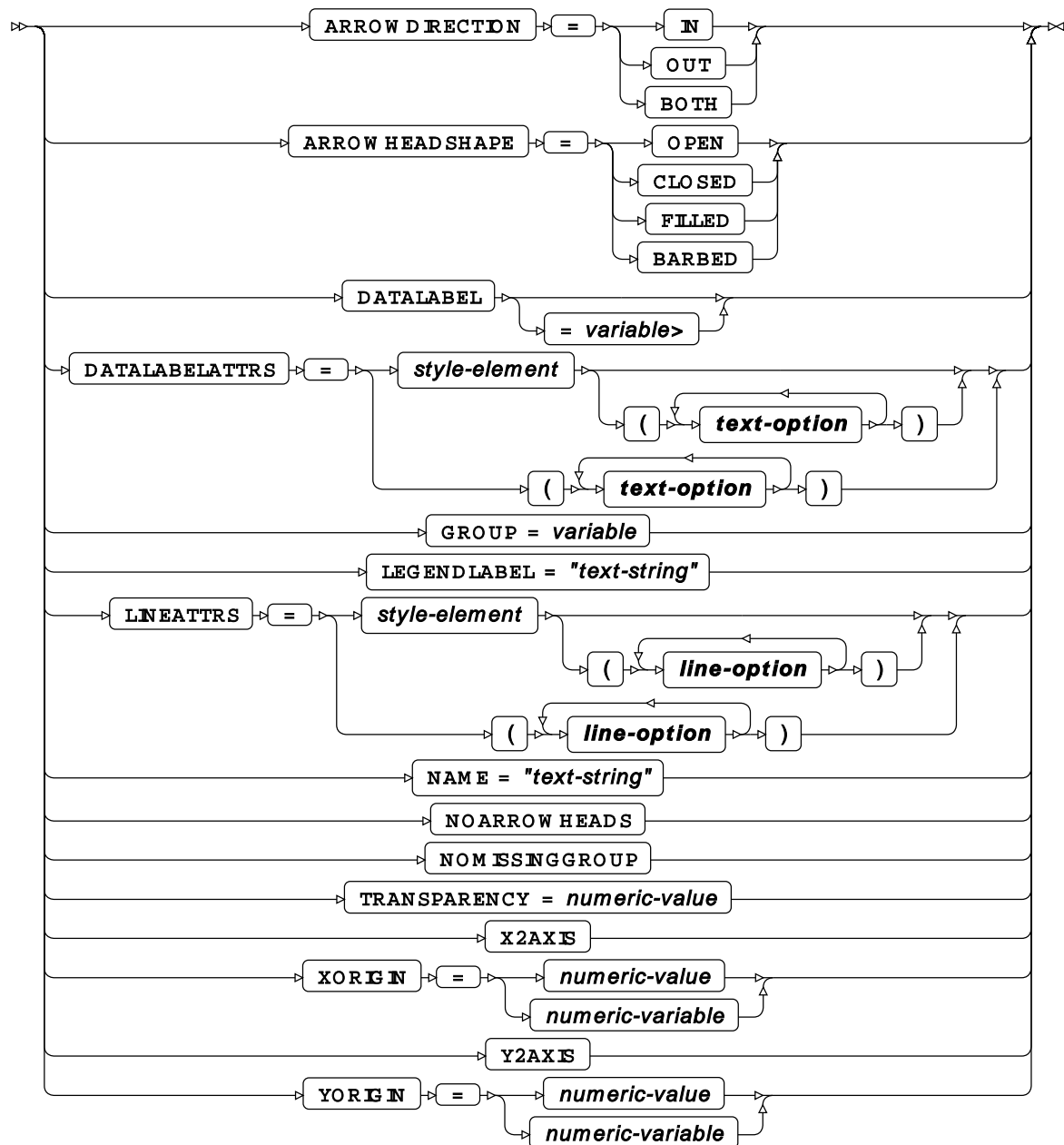


VECTOR

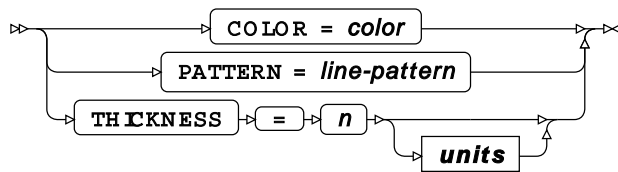
Draws a vector plot.



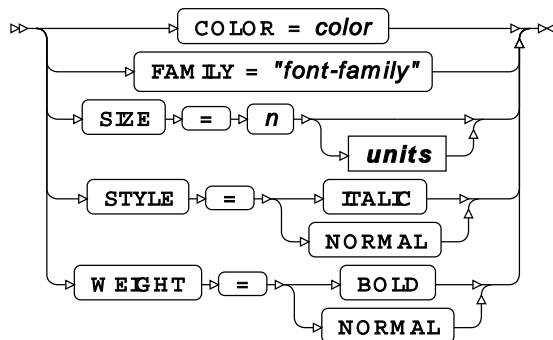
option



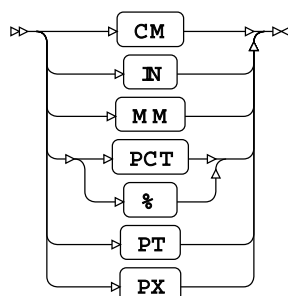
line-option



text-option

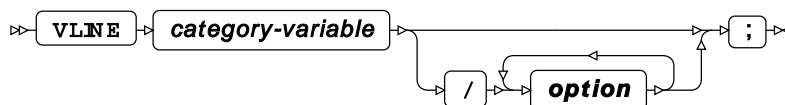


units

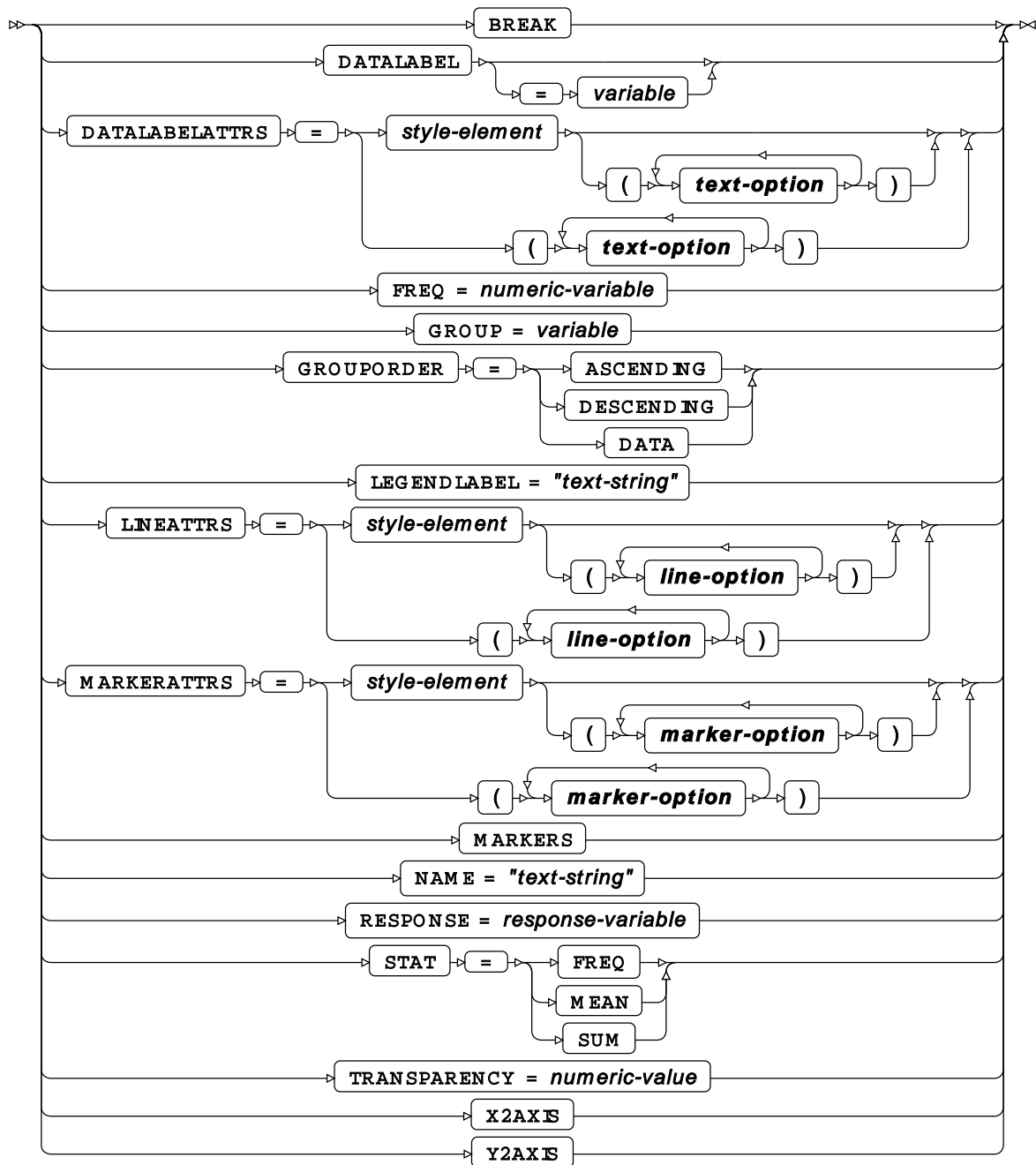


VLINE

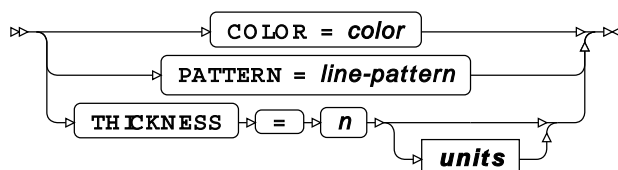
Draws a vertical line plot using unsummarised data.



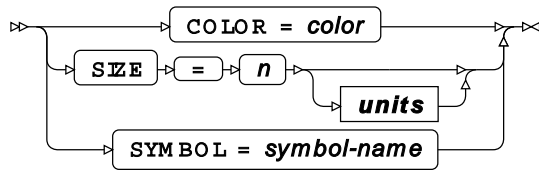
option



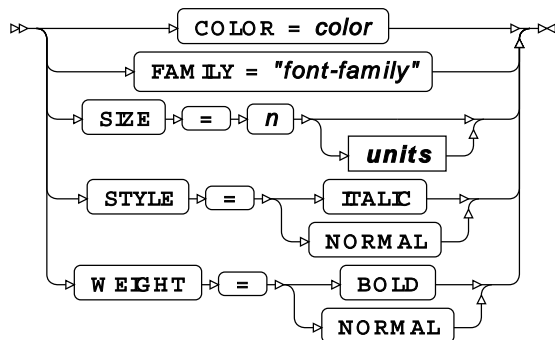
line-option



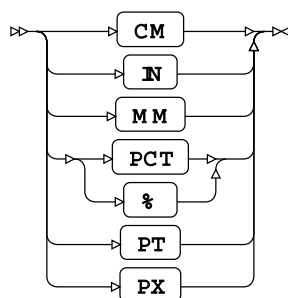
marker-option



text-option

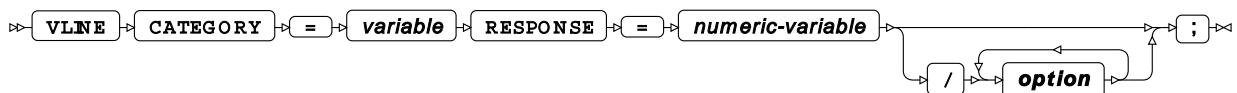


units

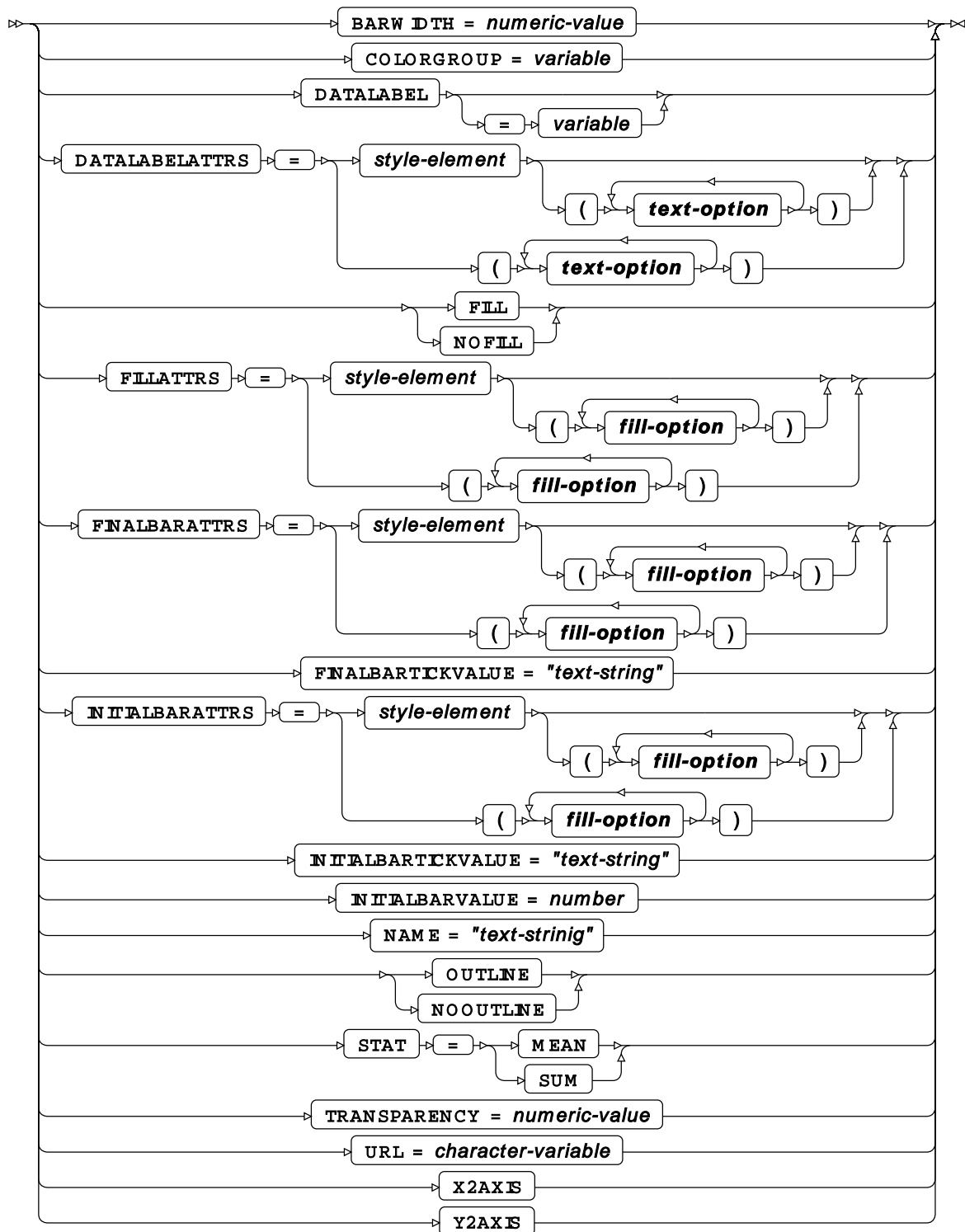


WATERFALL

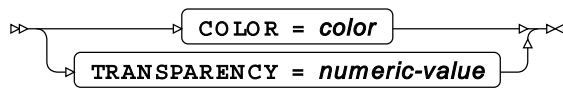
Draws a waterfall plot.



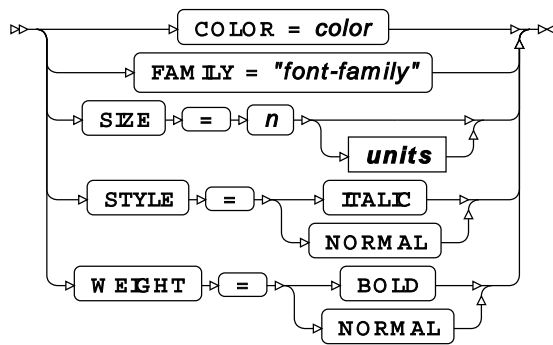
option



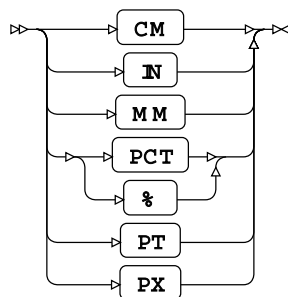
fill-option



text-option

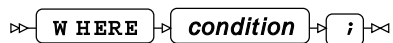


units



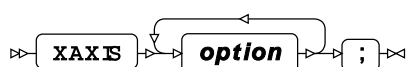
WHERE

Restricts the observations to be processed.

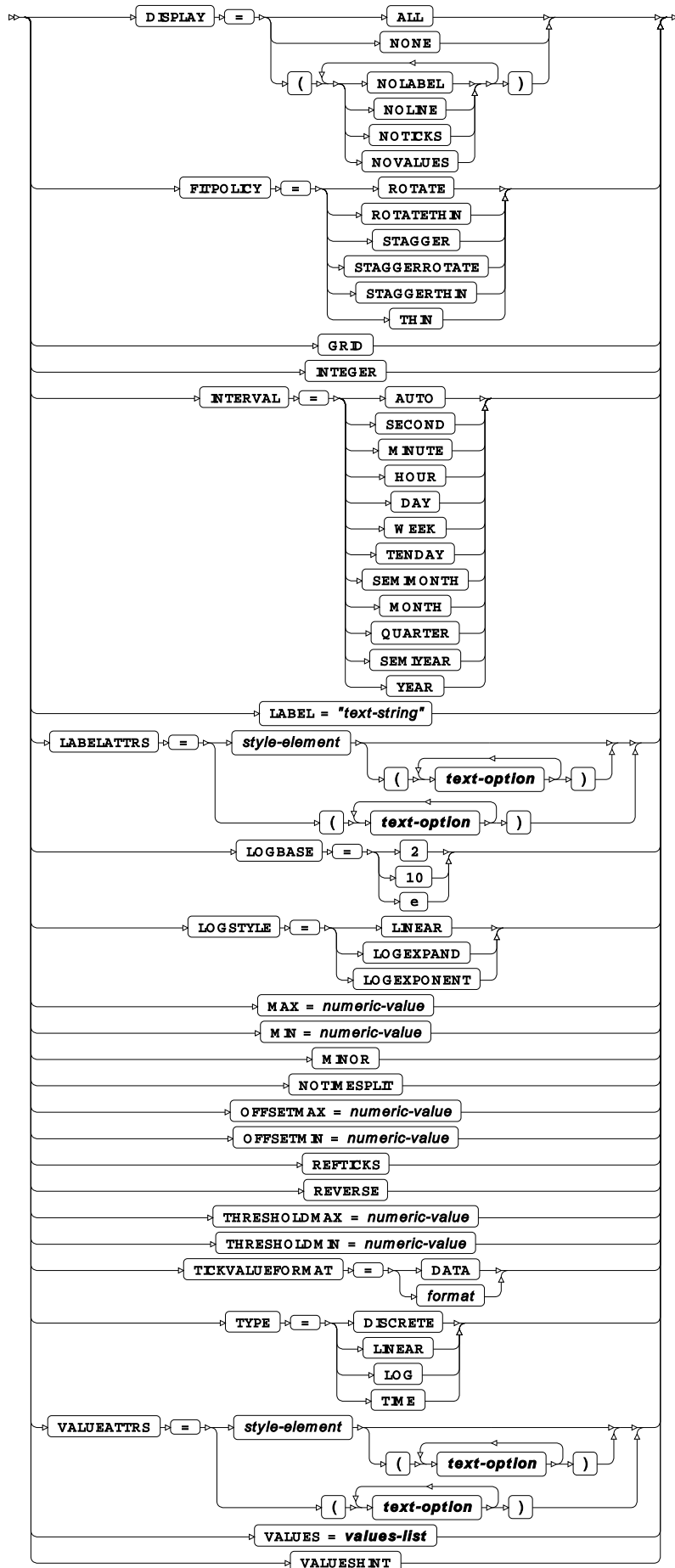


XAXIS

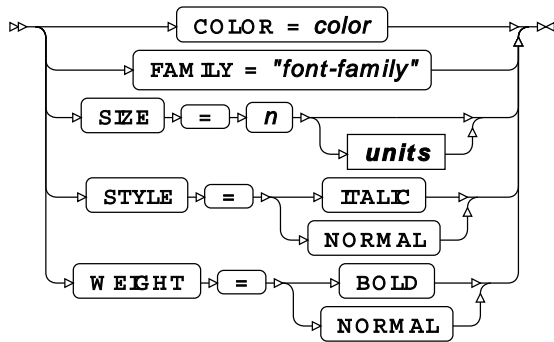
Specifies options for the primary X-axis



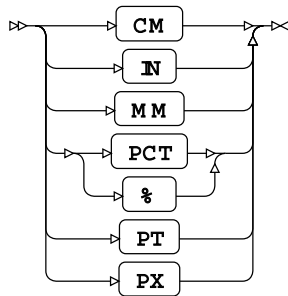
option



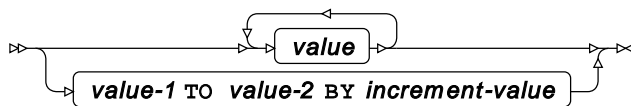
text-option



units

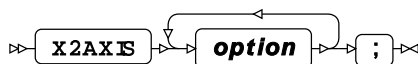


values-list

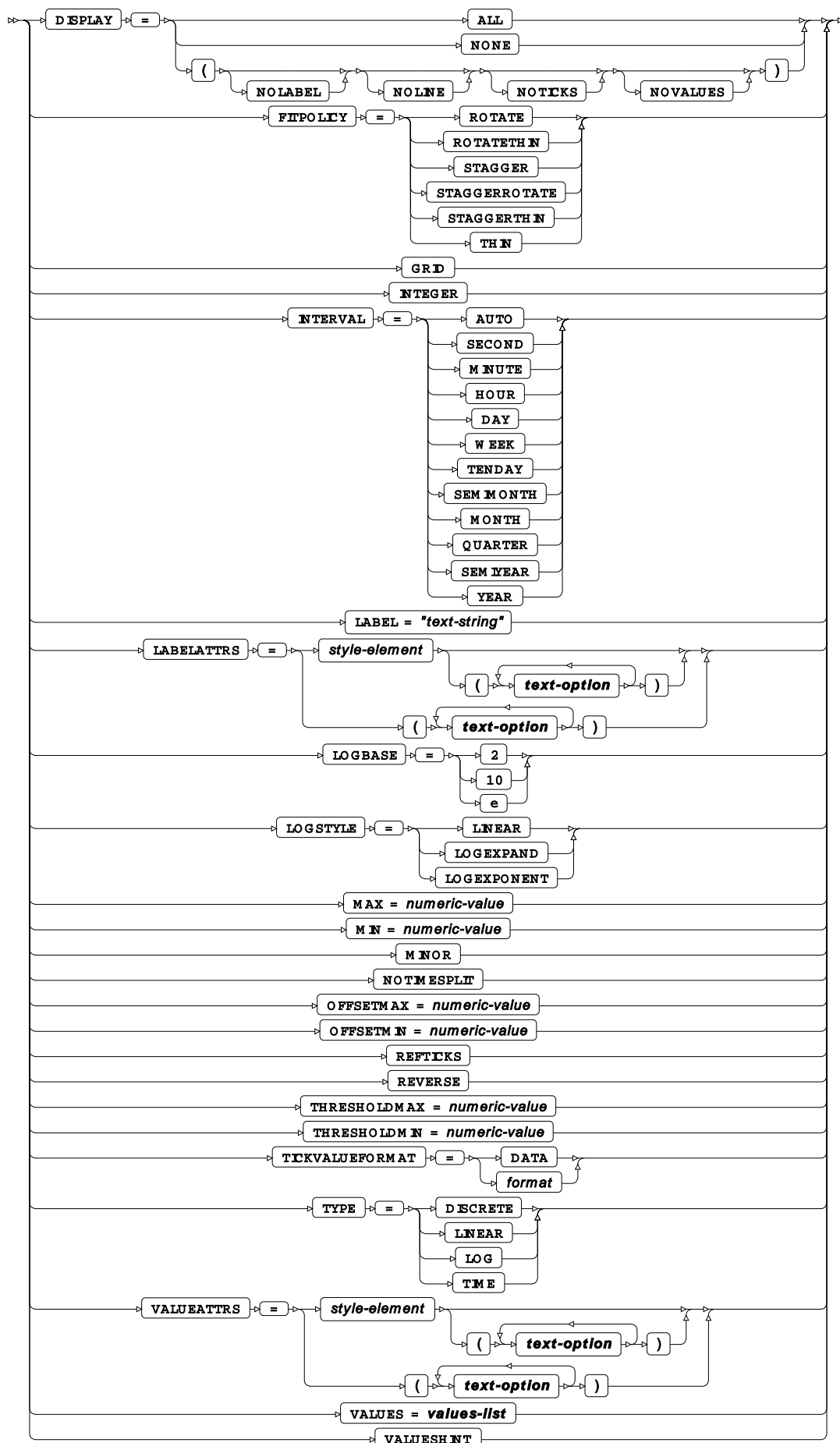


X2AXIS

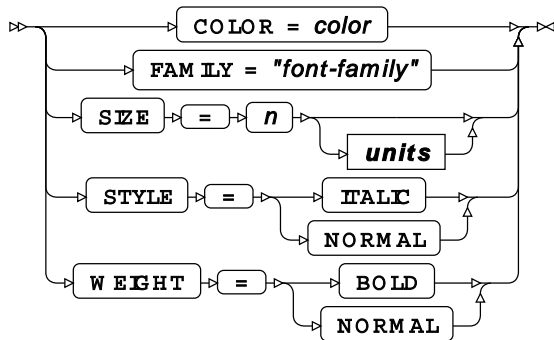
Specifies options for the secondary Y-axis



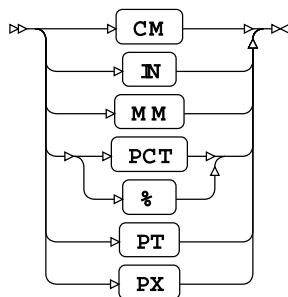
option



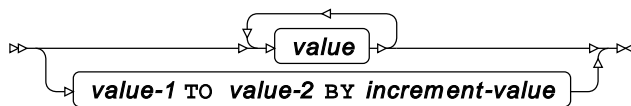
text-option



units

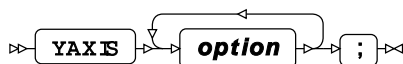


values-list

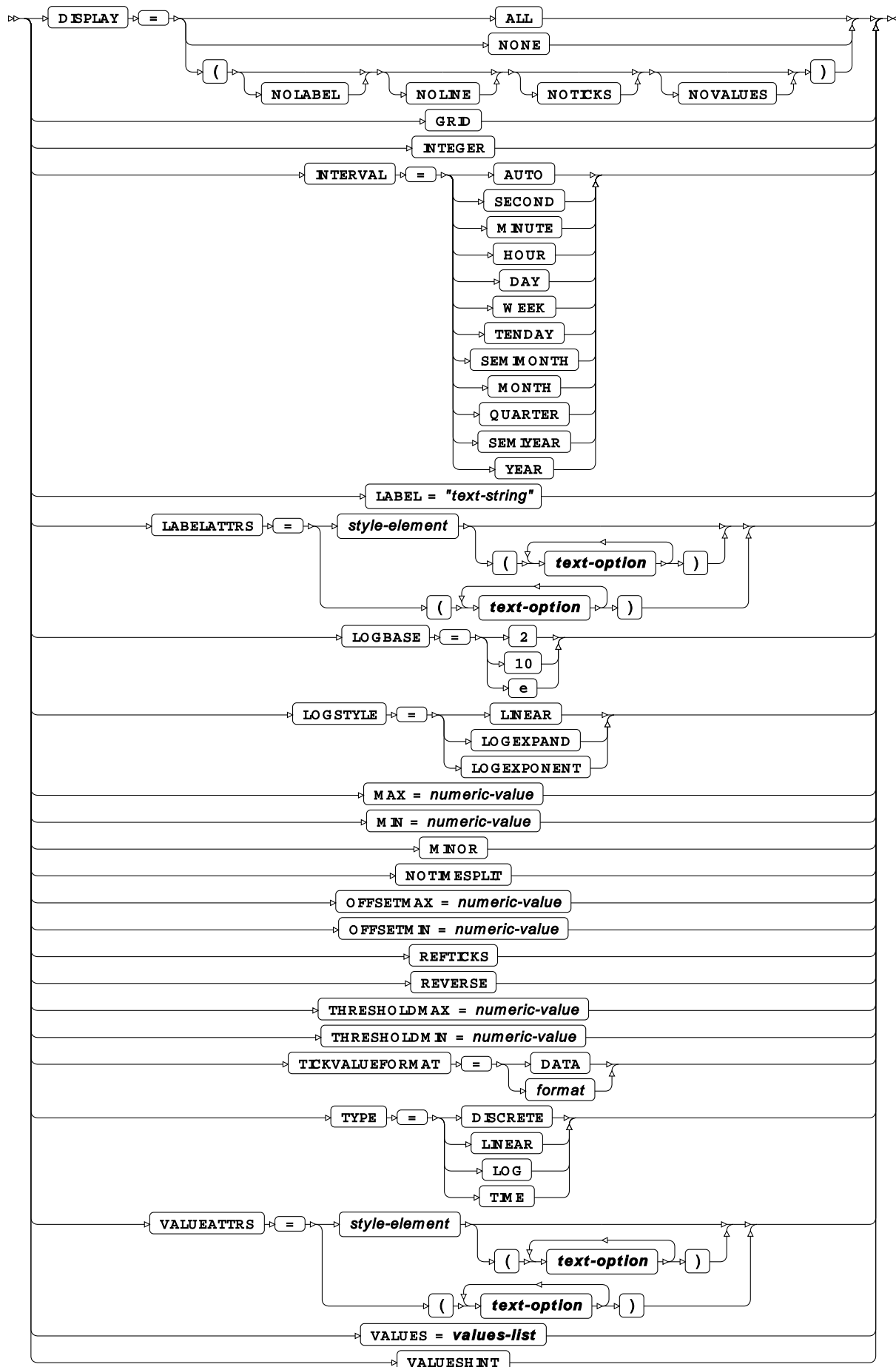


YAXIS

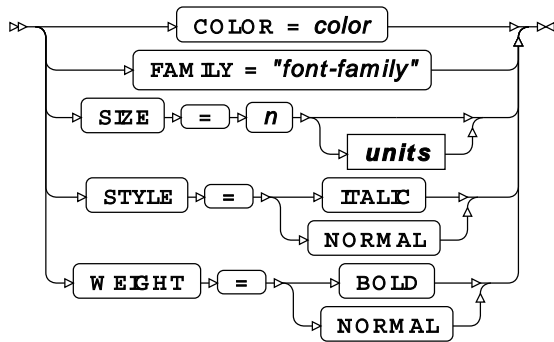
Specifies options for the primary Y-axis



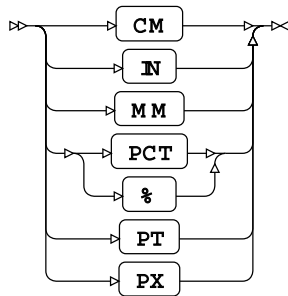
option



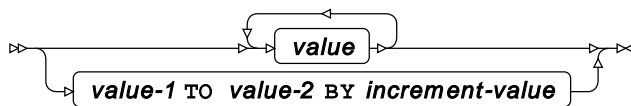
text-option



units

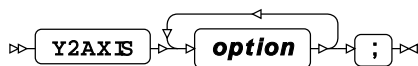


values-list

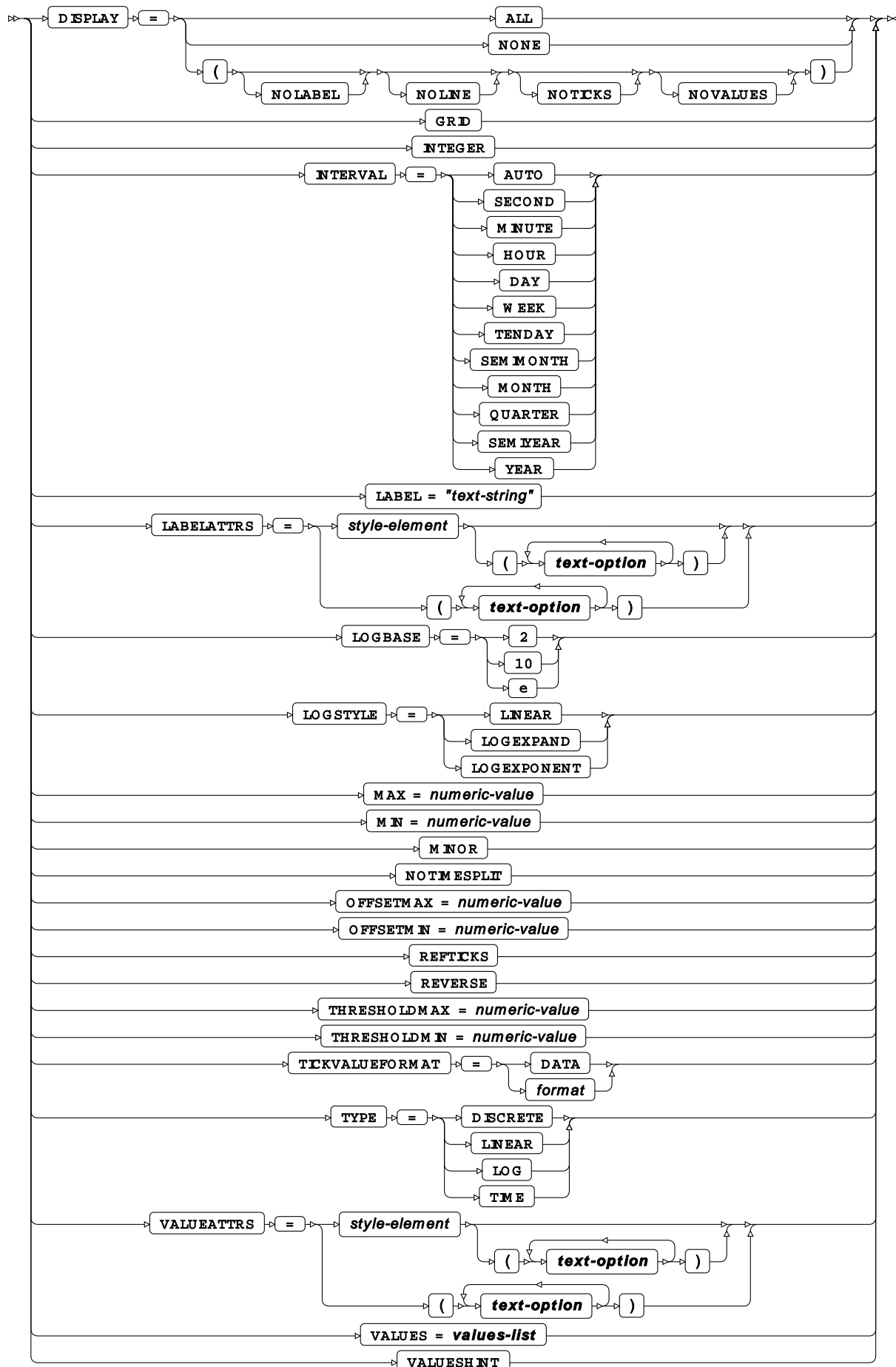


Y2AXIS

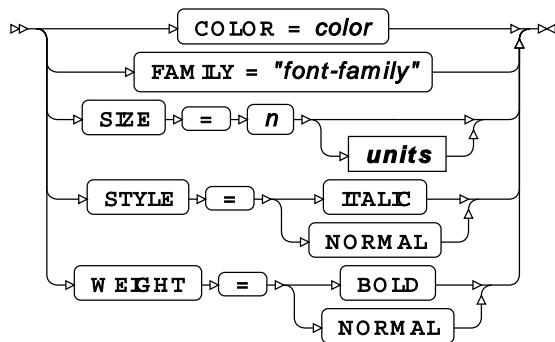
Specifies options for the secondary Y-axis



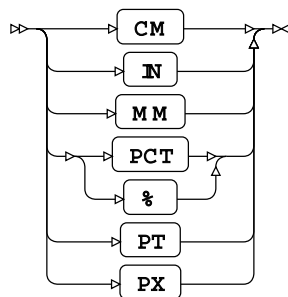
option



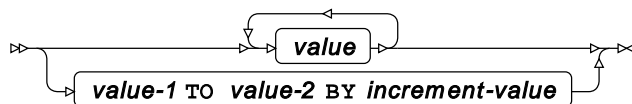
text-option



units



values-list



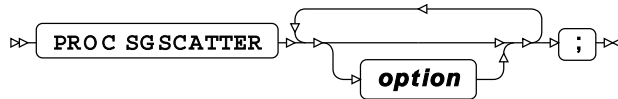
SGSCATTER procedure

Supported statements

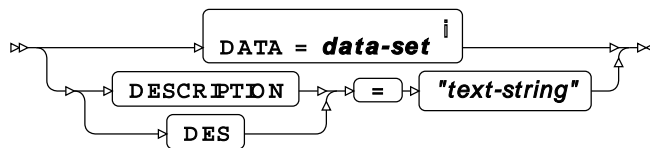
- **PROC SGSCATTER** [↗](#) (page 2778)
- **BY** [↗](#) (page 2778)
- **COMPARE** [↗](#) (page 2778)
- **MATRIX** [↗](#) (page 2782)
- **PLOT** [↗](#) (page 2784)
- **WHERE** [↗](#) (page 2788)

PROC SGSCATTER

Creates a grid of multiple scatter plots, where the number of plots are specified using one of the COMPARE, MATRIX, or PLOT statements.



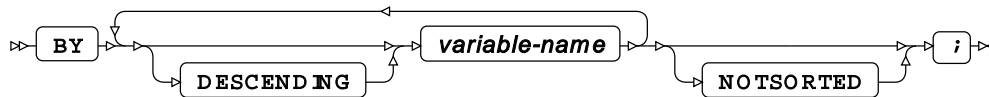
option



ⁱ See [Input dataset](#) (page 16).

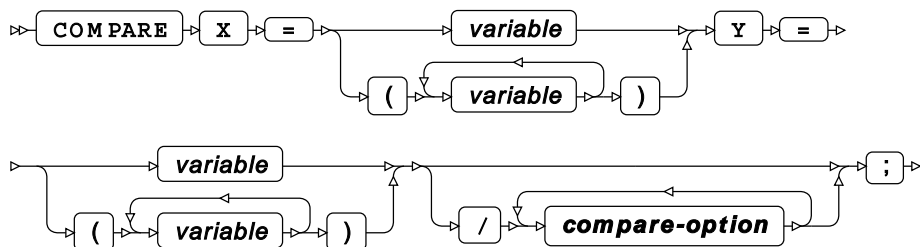
BY

Groups the observations in a dataset using one or more specified variables.

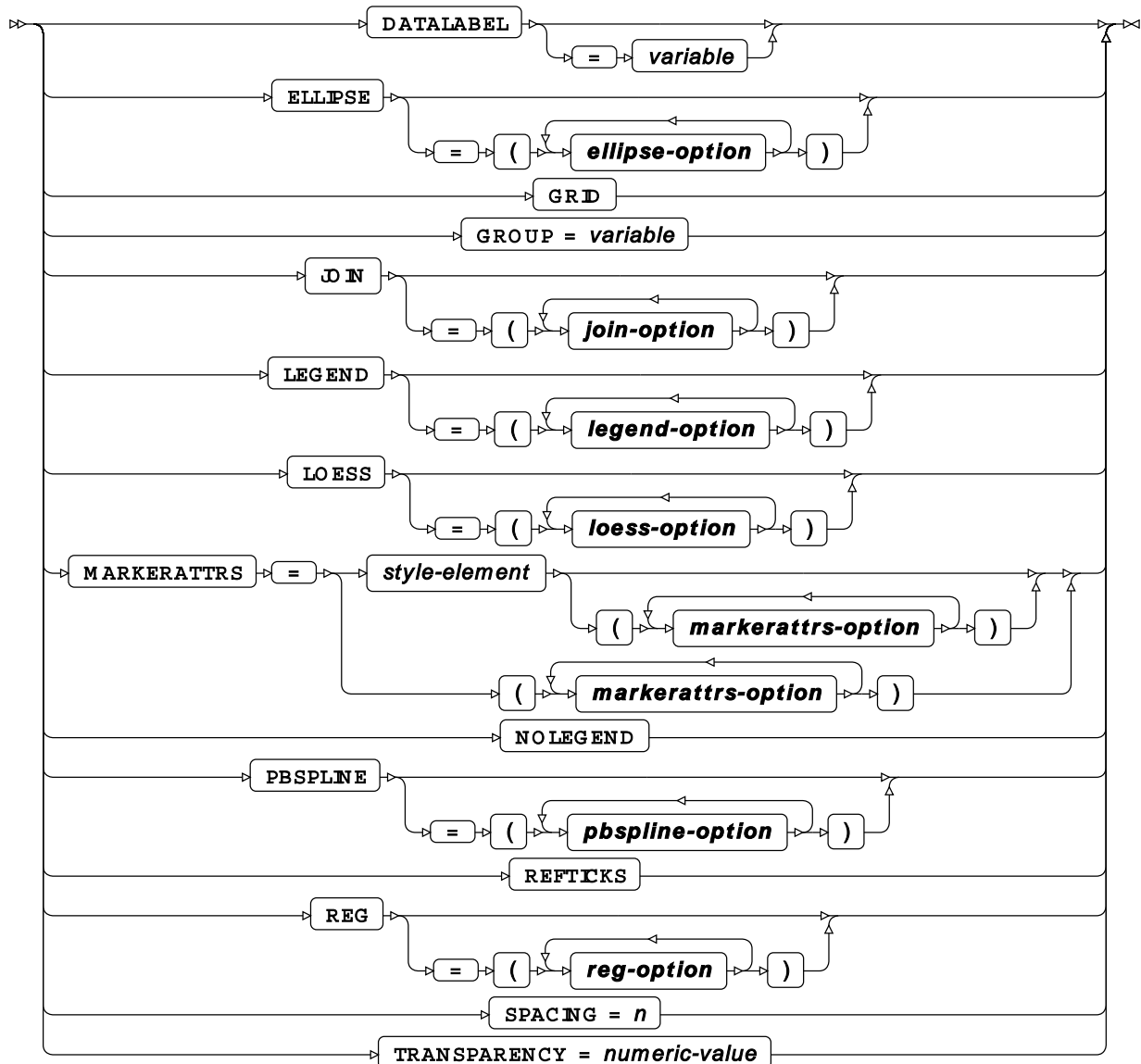


COMPARE

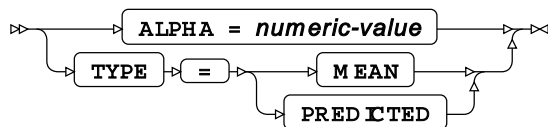
Creates a grid of scatter plots for the specified variables, where axes are shared for grid rows and grid columns.



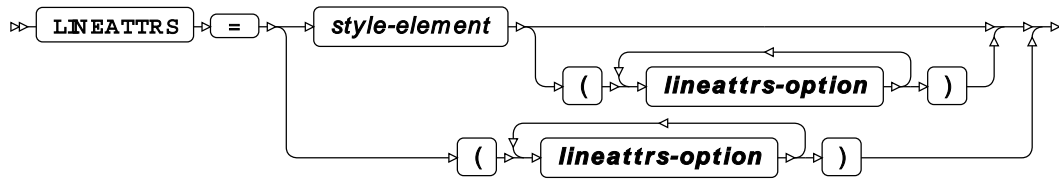
compare-option



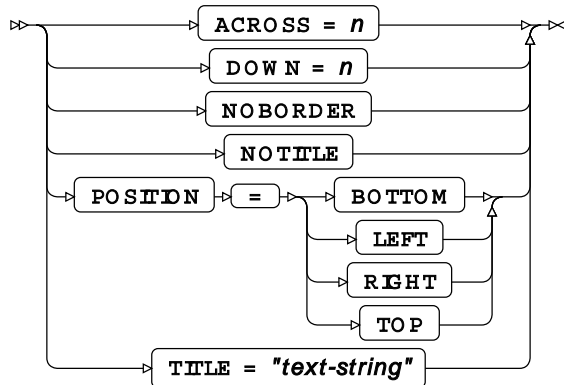
ellipse-option



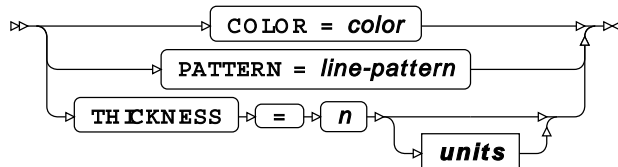
join-option



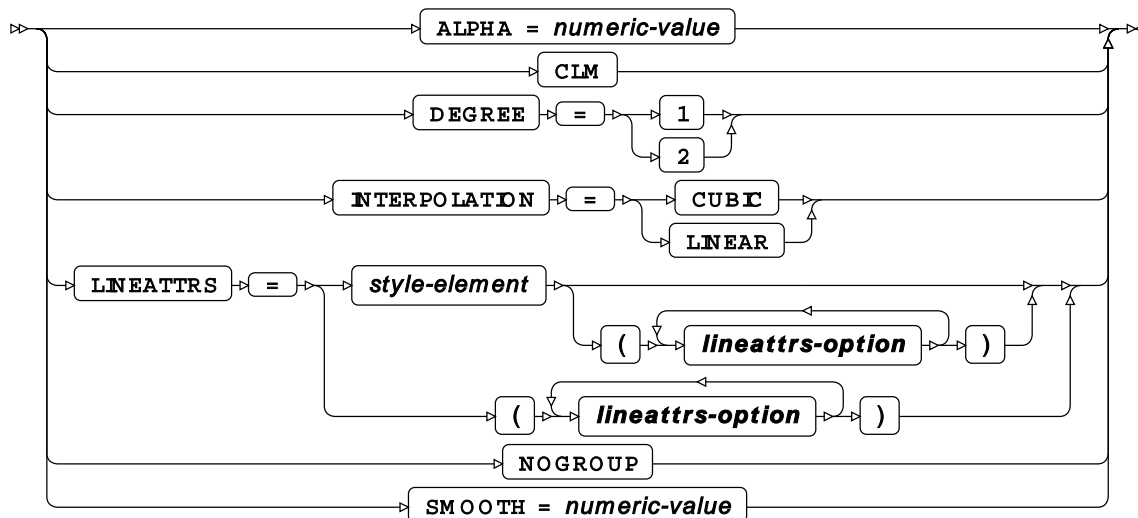
legend-option



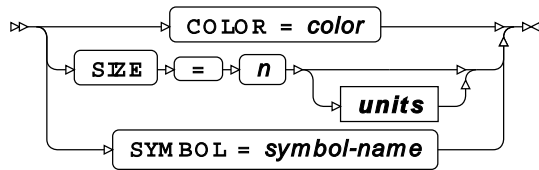
lineattrs-option



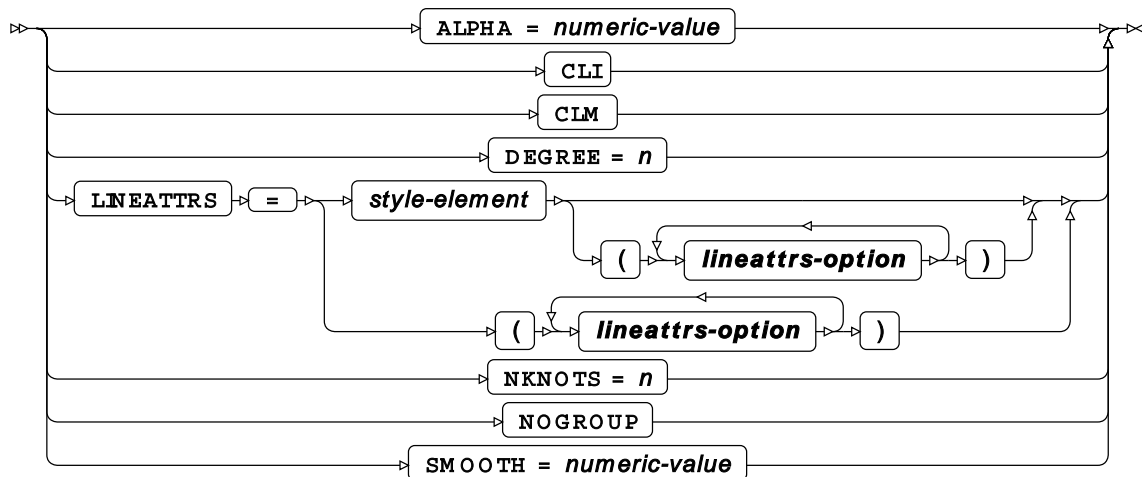
loess-option



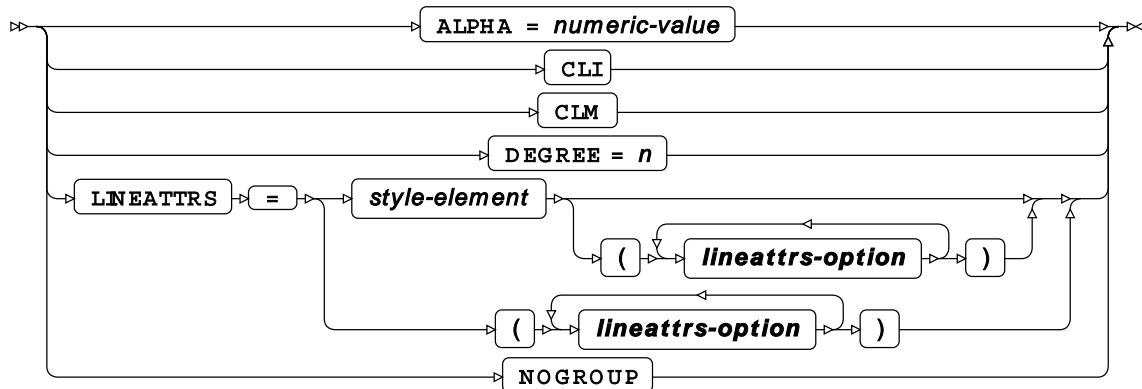
markerattrs-option



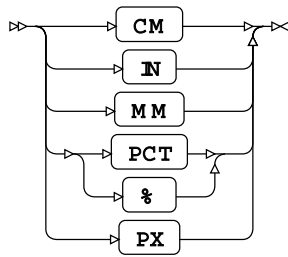
pbspline-option



reg-option

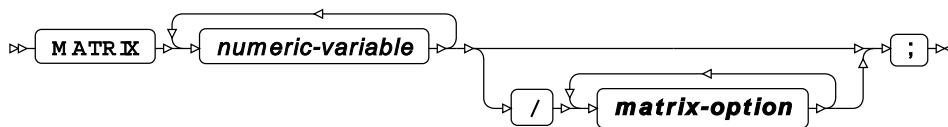


units

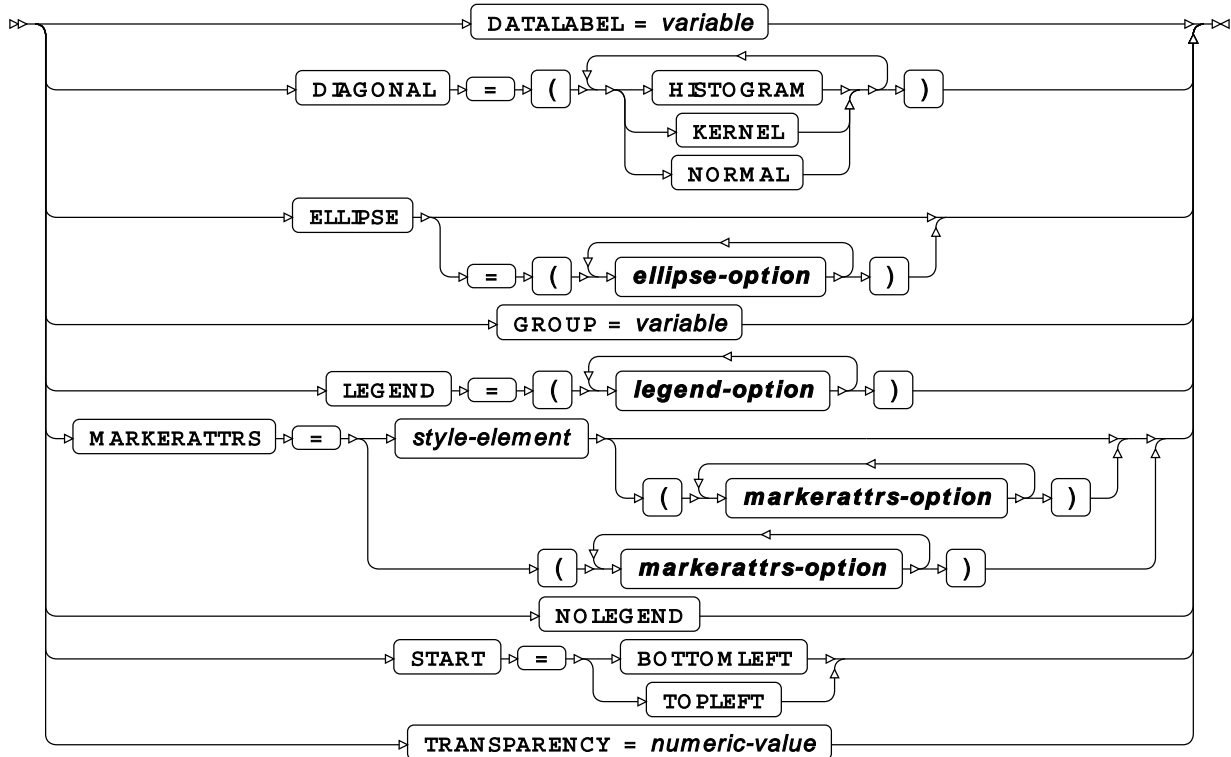


MATRIX

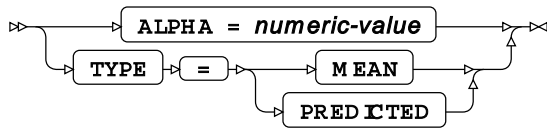
Creates a series of scatter plots for all possible pairs of the specified variables, where each plot shows one pair combination.



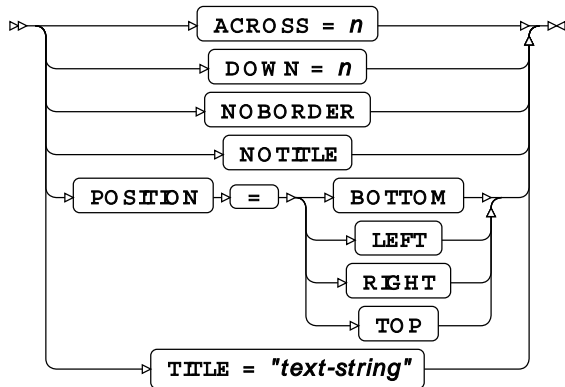
matrix-option



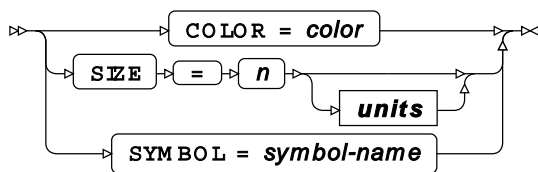
ellipse-option



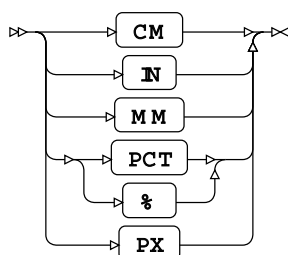
legend-option



markerattrs-option

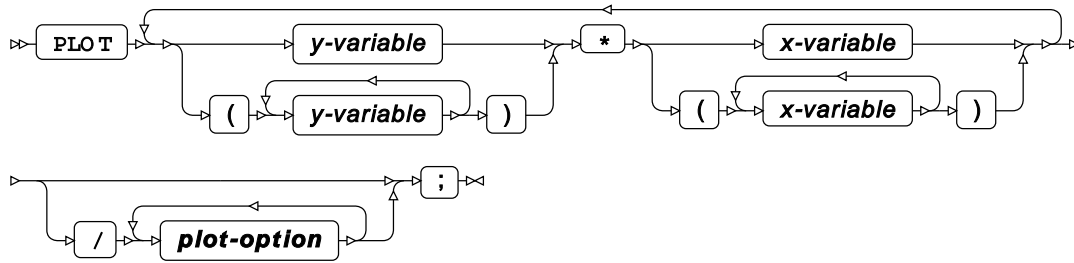


units

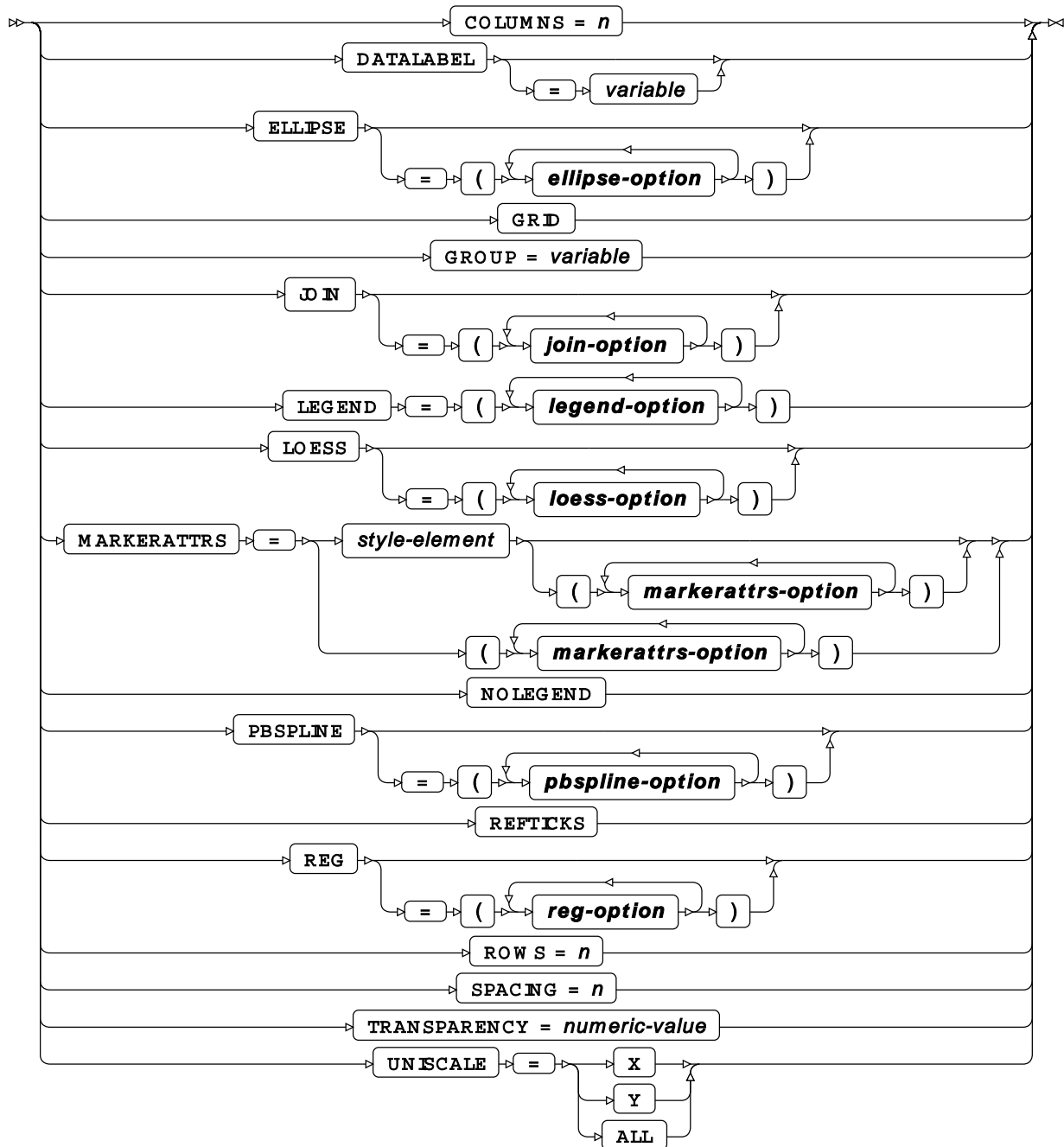


PLOT

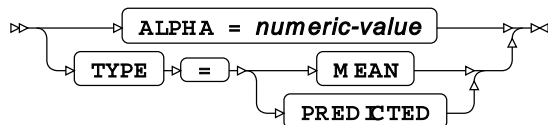
Creates a grid of scatter plots for the specified variables, where individual plots have their own axes.



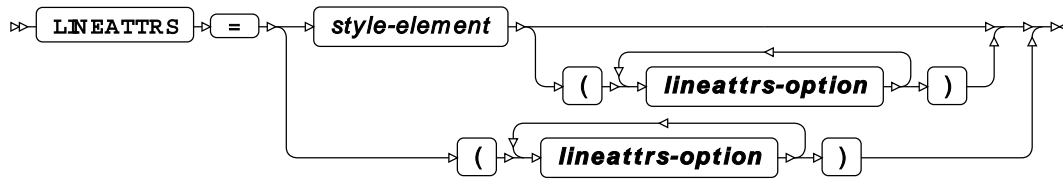
plot-option



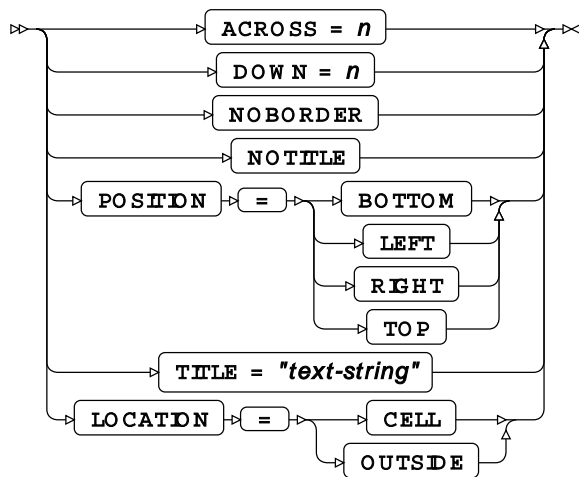
ellipse-option



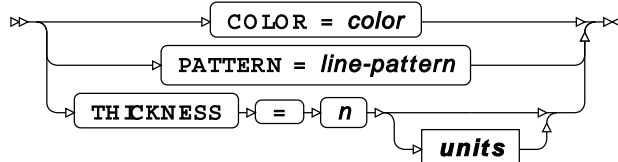
join-option



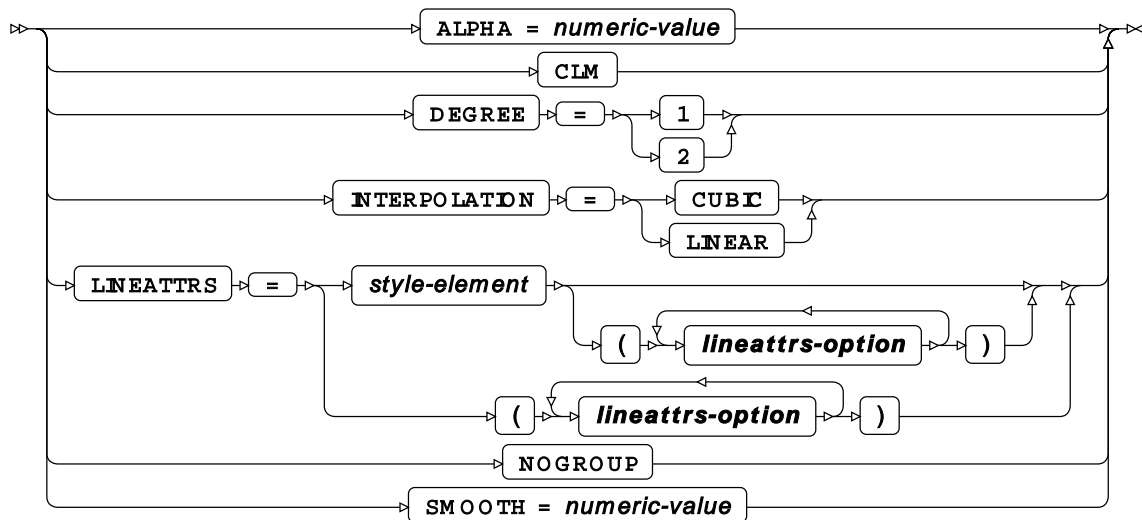
legend-option



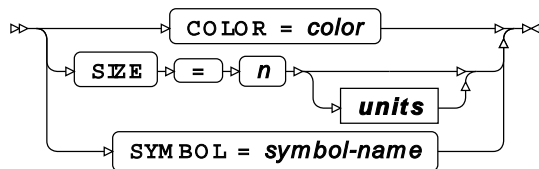
lineattrs-option



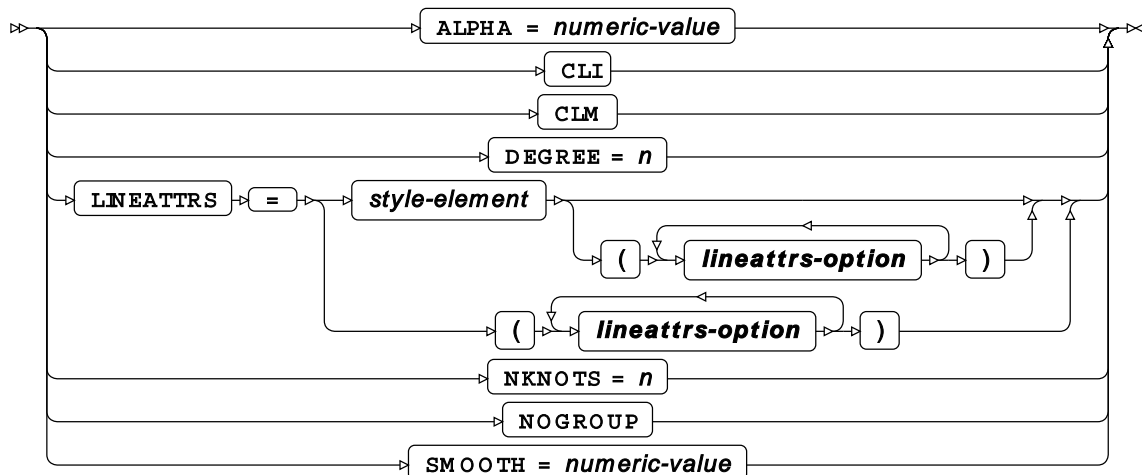
loess-option



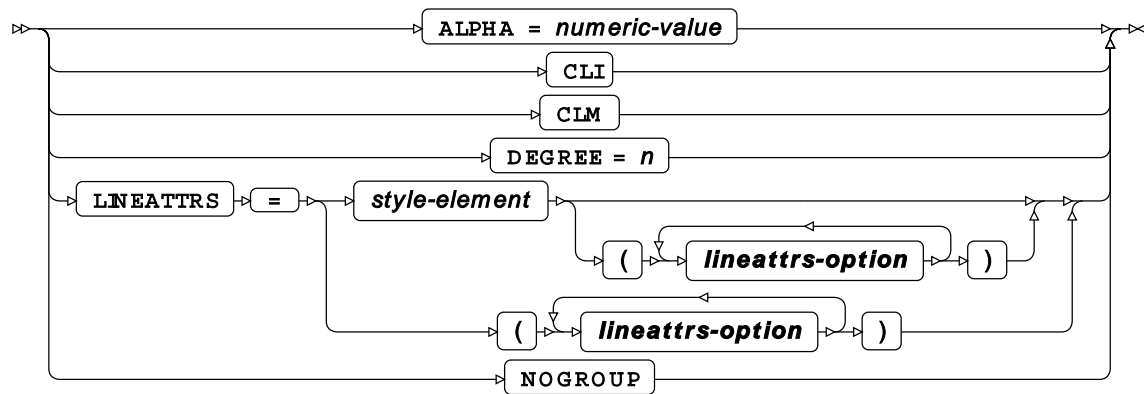
markerattrs-option



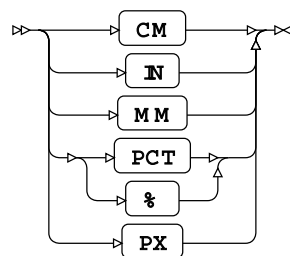
pbspline-option



reg-option

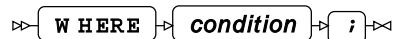


units



WHERE

Restricts the observations to be processed.



WPS Statistics

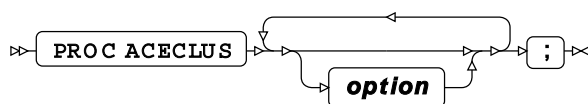
Statistics procedures

ACECLUS procedure

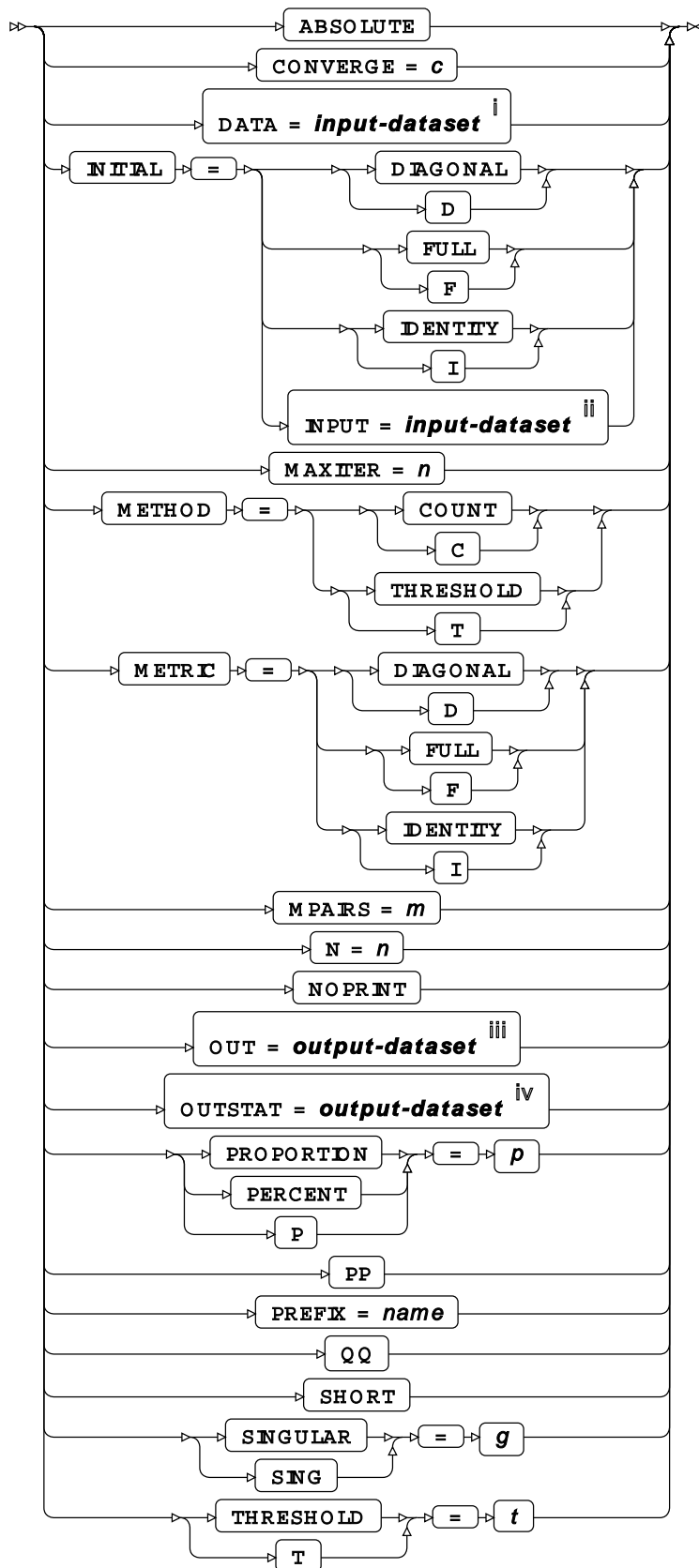
Supported statements

- *PROC ACECLUS* [↗](#) (page 2789)
- *ATTRIB* [↗](#) (page 2791)
- *BY* [↗](#) (page 2791)
- *FORMAT* [↗](#) (page 2791)
- *FREQ* [↗](#) (page 2791)
- *INFORMAT* [↗](#) (page 2792)
- *LABEL* [↗](#) (page 2792)
- *VAR* [↗](#) (page 2792)
- *WHERE* [↗](#) (page 2792)

PROC ACECLUS

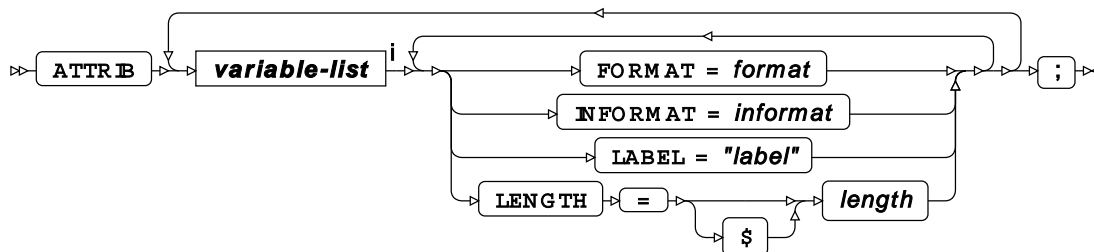


option



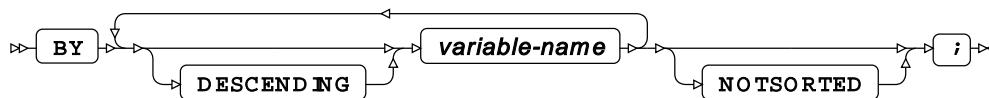
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).
- ^{iv} See *Input dataset* [↗](#) (page 16).

ATTRIB



- ⁱ See *Variable Lists* [↗](#) (page 32).

BY

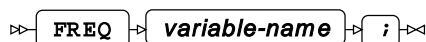


FORMAT

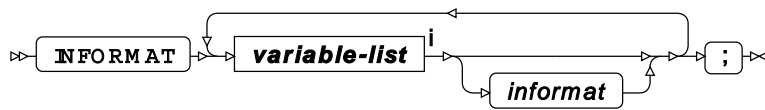


- ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

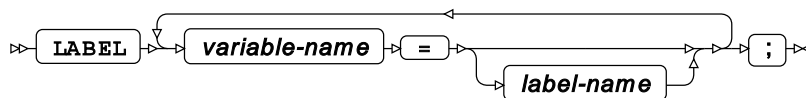


INFORMAT

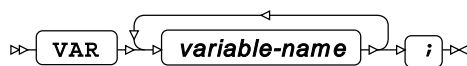


ⁱ See *Variable Lists* [↗](#) (page 32).

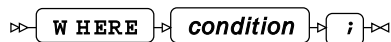
LABEL



VAR



WHERE



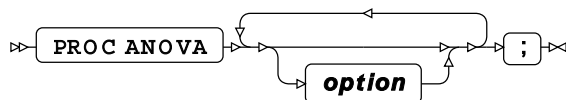
ANOVA procedure

Supported statements

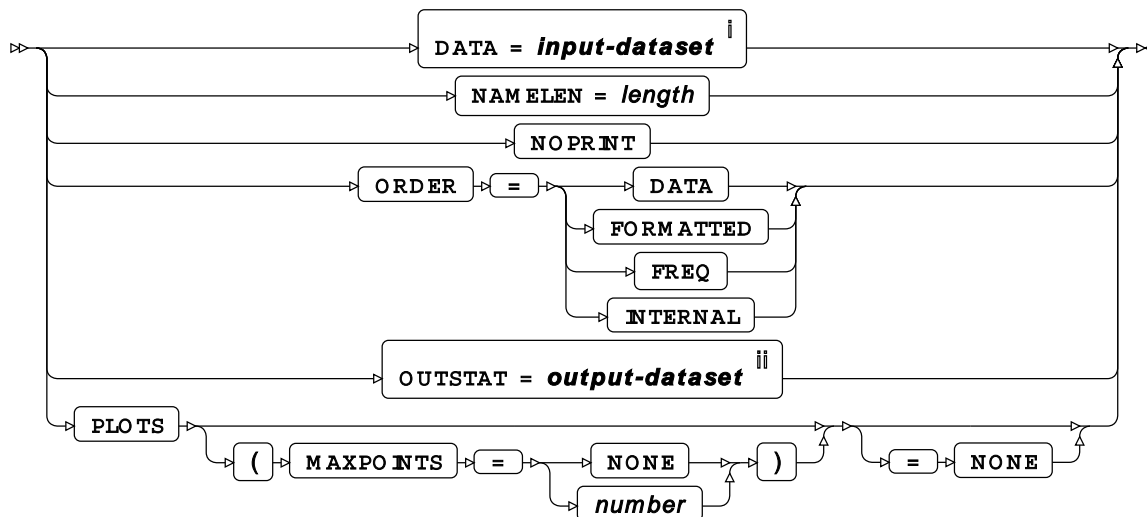
- *PROC ANOVA* [↗](#) (page 2793)
- *ATTRIB* [↗](#) (page 2793)
- *BY* [↗](#) (page 2794)
- *CLASS* [↗](#) (page 2794)
- *FORMAT* [↗](#) (page 2794)
- *FREQ* [↗](#) (page 2794)
- *INFORMAT* [↗](#) (page 2794)
- *LABEL* [↗](#) (page 2795)

- [MEANS](#) (page 2795)
- [MODEL](#) (page 2796)
- [TEST](#) (page 2796)
- [WHERE](#) (page 2796)

PROC ANOVA



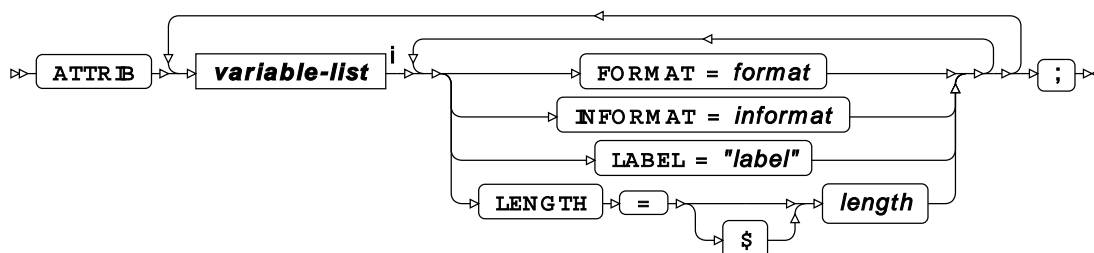
option



ⁱ See [Input dataset](#) (page 16).

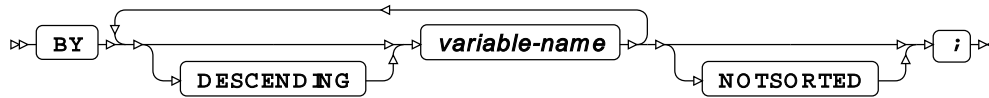
ⁱⁱ See [Input dataset](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

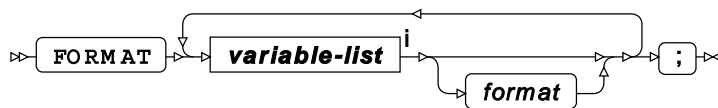
BY



CLASS

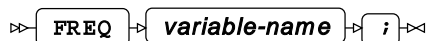


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

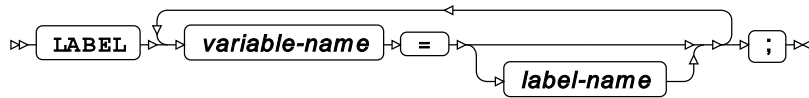


INFORMAT

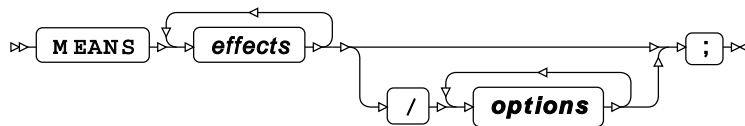


ⁱ See *Variable Lists* [↗](#) (page 32).

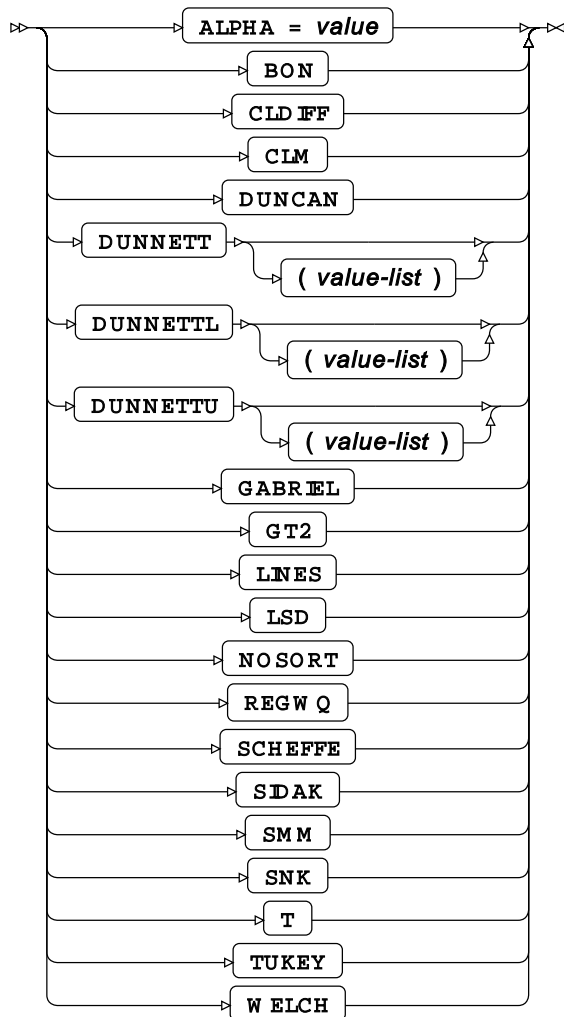
LABEL



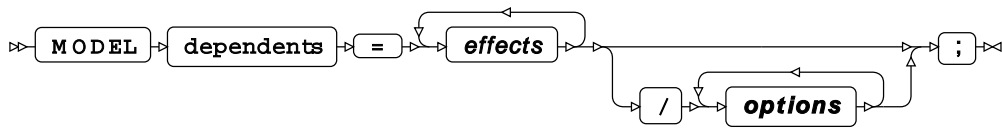
MEANS



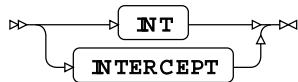
options



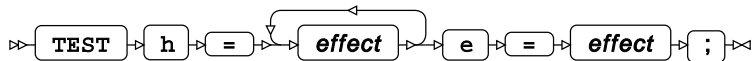
MODEL



options



TEST



WHERE



ASSOCRULES procedure

The ASSOCRULES procedure performs associative rule mining or associative rule matching on WPS datasets.

Rule mining is the default mode of operation, where ASSOCRULES takes an appropriately formatted dataset as an input, and outputs a list of association rules to ODS tables and/or a new dataset.

Rule matching is the other mode of operation for ASSOCRULES, where existing association rules are used from a prior invocation of the mining mode and applied to a new compatible dataset.

Overview of Association Rules [↗](#)..... 2797
Association rules describe co-occurrence relations between objects in a dataset.

About ASSOCRULES [↗](#)..... 2797
 ASSOCRULES can both generate association rules datasets and also use previously generated rules to report on matches against other datasets.

How to use ASSOCRULES ↗	2800
Two examples are provided, one for each mode of operation. The first example uses ASSOCRULES to mine association rules from a transaction dataset. The second example uses ASSOCRULES to generate matches from existing association rules and a new dataset.	
ASSOCRULES syntax ↗	2804
Describes the syntax and options for ASSOCRULES and its contained statements.	

Overview of Association Rules

Association rules describe co-occurrence relations between objects in a dataset.

For a dataset to be compatible with ASSOCRULES, it must consist of *items* grouped into *transactions*. Such datasets are often called *transaction datasets*. Items and transactions are abstract variables that can represent anything. For example, when analysing products bought in a shop, each transaction in a dataset could contain a set of multiple items sold to a customer in a single sale. Alternatively, a transaction dataset could contain a list of students and the subjects they have chosen to study; with each student being a transaction, and each subject being an item within those transactions. Both of these examples are given in *How to use ASSOCRULES* [↗](#) (page 2800).

Association rules for a transaction dataset describe co-occurrences within that dataset. For example, a rule could state that customers buying bread and milk are also likely to buy cereal, or that a student studying physics is also likely to study maths. The process of an algorithm discovering those rules is called *mining*.

Once association rules have been mined for a transaction dataset, they can be stored in a *rules dataset*, which can be matched in the future with new transaction datasets to ascertain how well that new dataset matches those existing association rules.

About ASSOCRULES

ASSOCRULES can both generate association rules datasets and also use previously generated rules to report on matches against other datasets.

Apriori Algorithm Implementations

ASSOCRULES can use two different implementations of the Apriori algorithm, as specified by the procedure's options: either the open source Borgelt or a WPS implementation. The WPS algorithm is included to enable future development of proprietary features.

Transaction datasets

ASSOCRULES requires datasets of transactions and items to only contain two types of variable: a transaction ID and an item ID. These variables can be character or numeric.

For example, the following raw data could be collected concerning sales of products in a shop:

Transaction ID	Milk	Cereal	Bread	Eggs	Crisps	Jelly	Sweets
1	1	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	0	0	0	1	1	1

For ASSOCRULES to process this raw data, it must be formatted into a two column dataset as follows:

Transaction ID	Item ID
1	Milk
1	Cereal
2	Bread
3	Crisps
3	Jelly
2	Milk
2	Eggs
3	Sweets

Entries in the dataset do not have to be grouped by transaction, although if they are then they will typically take less time to process.

The ordering of the individual items within a single transaction has no effect on the rules that ASSOCRULES finds.

Dataset entries with missing values are dropped before the dataset is processed; for example, if a Transaction ID is given without a corresponding item ID.

Rule Thresholds

To control the number of rules that are generated or used, and also to maximize their robustness and relevance, ASSOCRULES can reject rules that fail to meet user-specified thresholds for the following measures:

- *Support*: the proportion of transactions in a dataset in which an object appears. Support is expressed as a decimal value between 0 and 1.
- *Confidence*: the proportion of transactions in a dataset in which a rule linking two objects has been found to be true. Support is expressed as a decimal value between 0 and 1.
- *Lift*: the ratio of the observed support to that expected if the two linked objects were independent.

Use of system resources

Note:

The Apriori algorithm can use substantial system resources if the rule thresholds are set inappropriately and/or the input dataset is large. If ASSOCRULES runs out of memory, it reports an error and terminates. Under these circumstances, it is recommended that either the rule thresholds are changed, or the SAMPLEPROBABILITY option is used to analyse a randomly selected subset of the data.

Mining of association rules from a transaction dataset

By default, ASSOCRULES mines association rules from a transaction dataset. This mode of operation is employed if the `INEST` option is not specified.

Input

When mining association rules from a transaction dataset, ASSOCRULES accepts a transaction dataset as its input.

Output

If ASSOCRULES is configured to mine association rules from a transaction dataset, the following outputs can be specified:

- A sample of the rules generated is always written to the ODS tables. Options are available to set the size, content and order of this sample.
- The rules generated can be saved in a non human-readable format for subsequent use with ASSOCRULES in its other mode of operation: matching association rules to a transaction dataset.
- The rules generated can also be written to a dataset in human-readable form. The content and order of this sample can be set.

Matching association rules to a transaction dataset

If the `INEST` option is specified, ASSOCRULES reports matches between a transaction dataset and association rules it has previously generated in the rule mining mode of operation.

Input

To report on matches from association rules and a transaction dataset, ASSOCRULES requires:

- A dataset of rules previously generated by ASSOCRULES.
- A dataset of transactions in a format readable by ASSOCRULES.

Note:

The dataset of rules to be matched must have been generated from a dataset that is compatible with the new dataset being matched against. The two datasets are viewed as compatible if they have the same type and format of data (string or numeric) in each column.

Output

If ASSOCRULES is configured to generate matches between a previously generated dataset of rules and a compatible dataset of transactions, you can specify the following outputs:

- A sample of the data on matches generated is always written to the ODS output. Options are available to set the size, content and order of this sample.
- A full set of match data can be written to a dataset in human-readable form. The content and order of this output can be set.

How to use ASSOCRULES

Two examples are provided, one for each mode of operation. The first example uses ASSOCRULES to mine association rules from a transaction dataset. The second example uses ASSOCRULES to generate matches from existing association rules and a new dataset.

Mining association rules from a transaction dataset

This example uses ASSOCRULES to mine association rules from a transaction dataset. This is the default mode of operation if the INEST option is not specified.

This example is available as 13.0-AssocrulesMining.sas in the samples folder distributed with WPS Analytics.

Transaction dataset

In this example, the ASSOCRULES procedure uses a simple dataset of nine transactions from a shop. The first variable contains numerical transaction identifiers from 1 to 9. The second variable contains items purchased. Part of the dataset is reproduced below:

Transaction	Item
1	Cereal
2	Bread
2	Milk
2	Eggs
...	
7	Eggs
8	Bacon
8	Sausages
9	Milk

The above dataset is stored in a WPS workspace named TransactionData1.

Example Code for the ASSOCRULES procedure

The SAS language code for the ASSOCRULES procedure in this example is:

```
PROC ASSOCRULES OUTEST=transoplib DATA=TransactionData1 MODE=WPL MINSUPPORT=1
  MINLIFT=4.0 RULESTOPRINT=10 SORTBY=(LIFT) PRINTCONFIDENCE PRINTLIFT;
  OUTPUT OUT=outputlib;
RUN;
```

Association rules are generated from the dataset `DATA=TransactionData1` and saved to a dataset named `transoplib`. Rules are generated using WPS' own implementation of the Apriori algorithm. Rules will only be reported with a support greater than or equal to 1, and a lift greater than or equal to 4. A maximum of 10 rules will be output to the ODS rules table. The output will include confidence and lift, and will be sorted by lift.

Output ODS tables

The above dataset and code produces three ODS tables:

Configuration	
Setting	Value
Minimum support threshold	1
Lift threshold	4

Results summary	
Result	Value
Unique items	10
Total observations	22
Observations dropped	0
Input transactions	9
Rules	33

Association rules			
Antecedent	Consequent	Confidence	Lift
Bacon Bread Milk	Eggs	1.000000	4.500000
Jelly	Crisps	1.000000	4.500000
Crisps Sweets	Jelly	1.000000	4.500000
Balloons Crisps Sweets	Jelly	1.000000	4.500000
Balloons Jelly	Crisps	1.000000	4.500000
Sweets	Crisps	1.000000	4.500000
Balloons Crisps	Sweets	0.500000	4.500000
Crisps	Balloons	1.000000	4.500000
Balloons Jelly	Sweets	0.500000	4.500000
Balloons	Sweets	0.500000	4.500000

Output dataset

This example also produces an output dataset, `outputlib`. This dataset contains numbered columns for each antecedent, followed by a column containing the consequent, and then details of: item set support, body set support, confidence and lift.

Matching association rules to a transaction dataset

This example uses `ASSOCRULES` to generate matches from a previously saved association rules dataset and a new transaction dataset. This mode of operation is invoked with the `INEST` option.

This example is available as `13.1-AssocrulesMatching.sas` in the samples folder distributed with WPS Analytics.

Preparation: Rule mining with a 2017 transaction dataset

This example uses a dataset of association rules produced from a simple transaction dataset of ten students joining a college to study A levels in 2017. Part of the 2017 dataset is reproduced below:

Student	Subject
John	Physics
John	Maths
John	ComputerScience
Jane	English
...	
Jasper	Physics
Julian	Maths
Julian	Physics
Julian	Chemistry

The `ASSOCRULES` procedure is used in a similar way to the mining example `13.0-AssocrulesMining.sas`. Rules generated are stored as a dataset named `alevelR`.

Rule matching with a 2018 Transaction Dataset

The `alevelR` rules dataset is applied to a new transaction dataset of students and subjects from 2018, named `ALevels2018`, part of which is reproduced below:

```
Student Subject
Simon   Physics
Simon   Maths
Simon   ComputerScience
Sophie  English

...

Steve   Physics
Susan   Maths
Susan   Physics
Susan   Chemistry
```

Details of the matches found are stored in the dataset `AlevelM2018`.

Example Code for the ASSOCRULES procedure

The SAS language code for the ASSOCRULES procedure in this example is:

```
PROC ASSOCRULES DATA=ALevels2018 PRINTCONFIDENCE INEST=alevelR;
    OUTPUT OUT=AlevelM2018;
RUN;
```

The presence of the `INEST` option instructs ASSOCRULES to run in rule matching mode, in this case with a rules dataset called `alevelR`, matching against the transaction dataset `ALevels2018`. The output includes confidence for each rule and is written to the dataset `AlevelM2018`.

Output ODS tables

The above dataset and code will produce two ODS tables as follows:

Configuration	
Setting	Value
Minimum support threshold	1

Match summary	
Result	Value
Rules	14
Total observations	27
Observations dropped	0
Input transactions	10
Matches reported	10

Output dataset

This example also produces an output dataset, `AlevelM2018`. This dataset contains columns for transactions and consequents, followed by details of: item set support, body set support, confidence and lift.

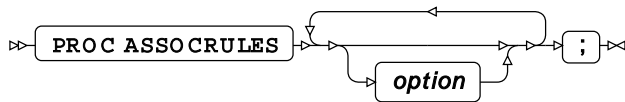
ASSOCRULES syntax

Describes the syntax and options for `ASSOCRULES` and its contained statements.

<code>PROC ASSOCRULES</code> ↗	2804
Describes the options for the <code>ASSOCRULES</code> procedure.	
<code>OUTPUT</code> ↗	2809
Describes the options for the <code>OUTPUT</code> statement available with the <code>ASSOCRULES</code> procedure.	

PROC ASSOCRULES

Describes the options for the `ASSOCRULES` procedure.



Options

The following *options* are available:

ANTECEDENTITEMSTOPRINT

Specifies the maximum number of antecedent items to print in the rules ODS table.

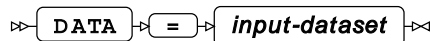


If this option is not specified, it will default to five. Rules with more than this specified number of items in their antecedents will have their antecedents truncated with an ellipsis at this length.

Must be greater than 1. A decimal value is truncated.

DATA

Specifies the dataset to be mined for association rules or to be matched against existing rules.



If the `INEST` option is not specified, `DATA` specifies the dataset to be mined for association rules.

If the `INEST` option is specified, association rules in the `INEST` dataset are matched to the transactions in the `DATA` dataset specified with the `ASSOCULES` procedure. The results are written to the `OUTPUT` dataset.

A `DATA` dataset specified with the `ASSOCULES` procedure must consist of two columns: the first must contain a unique identifier for every transaction in the dataset; and the second must provide item identifiers. Either column can be comprised of numbers or strings.

INEST

Configures `ASSOCRULES` to generate a dataset of matches from previously created association rules and specifies the dataset containing those association rules. If omitted, `ASSOCRULES` generates association rules from a dataset.

➤ `INEST` ➤ `=` ➤ `dataset` ➤

The rules in the specified dataset are matched to the transactions in the `DATA` dataset, which must be also specified. The results can be written to a dataset specified in the `OUTPUT` statement.

If you specify `INEST`, the input dataset specified by the `DATA` option must be in the same format as the dataset used to generate the rules in the `INEST` rules dataset. If this is not the case, WPS generates an error.

MAXITEMSPERRULE

Specifies the maximum number of items that can appear in a rule, or are required for matched rules to be reported.

➤ `MAXITEMSPERRULE` ➤ `=` ➤ `number` ➤

If the `INEST` option is not specified, `MAXITEMSPERRULE` specifies the maximum number of items that can appear in a rule.

If the `INEST` option is specified, `MAXITEMSPERRULE` specifies the maximum number of items required for matched rules to be reported.

Must be greater than or equal to 1. A decimal value is truncated.

MINCONFIDENCE

Specifies the minimum level of confidence that a rule requires to be reported, or that is required to report matches.

➤ `MINCONFIDENCE` ➤ `=` ➤ `number` ➤

If `INEST` is not specified, `MINCONFIDENCE` specifies the level of confidence that a rule must have to be reported.

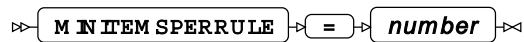
If `INEST` is specified, `MINCONFIDENCE` specifies the minimum confidence level required to report matches.

If this option is not specified, the default is 0.5.

Must be between 0 (zero) and 1.

MINITEMSPERRULE

Specifies the minimum number of items that can appear in a rule, or that are required in a rule for its matches to be reported.



If the `INEST` option is not specified, `MINITEMSPERRULE` specifies the minimum number of items that can appear in a rule.

If the `INEST` option is specified, `MINITEMSPERRULE` specifies the minimum number of items required in a rule for matches to be reported.

Must be greater than or equal to 1. A decimal value is truncated.

MINLIFT

Specifies the level of lift that a rule, or matches, requires to be reported.



If the `INEST` option is not specified, `MINLIFT` specifies the level of lift that a rule must achieve to be reported.

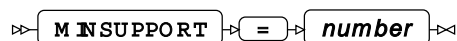
If the `INEST` option is specified, `MINLIFT` specifies the minimum lift required in a rule for matches to be reported.

If this option is not specified, the default is 1.1.

Must be greater than 0 (zero).

MINSUPPORT

Specifies the support required for a rule or its matches to be reported.



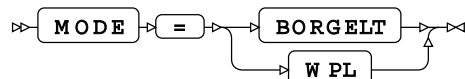
If `INEST` is not specified, `MINSUPPORT` is mandatory and specifies how much support a rule must have for it to be reported.

If `INEST` is specified, `MINSUPPORT` specifies the minimum support required for a rule for matches to be reported.

Must be greater than 0 (zero).

MODE

Specifies which rule mining implementation to use.



`BORGELT`, the default option, offers lower memory consumption and higher speed.

`WPL` is WPS Analytics' own algorithm, included to allow for future development of proprietary features.

OUTEST

Specifies a WPS dataset to be populated with a machine readable representation of the association rules found by the procedure.

» **OUTEST** = *dataset* «

The dataset can subsequently be loaded using the **INEST** option for rule matching against new transaction data.

To generate a human-readable representation of the association rules, see the **OUTPUT** option.

The **OUTEST** option is not compatible with the **INEST** option. If the two are specified together, an error is returned.

PRINTCONFIDENCE

Specifies that the rules table written to ODS output should include a confidence figure.

» **PRINTCONFIDENCE** «

PRINTBODYSETSUPPORT

Specifies that the rules table written to ODS output should include body set support.

» **PRINTBODYSETSUPPORT** «

PRINTITEMSETSUPPORT

Specifies that the rules table written to ODS output should include item set support.

» **PRINTITEMSETSUPPORT** «

PRINTLIFT

Specifies that the rules table written to ODS output should include lift.

» **PRINTLIFT** «

RULESTOPRINT

Specifies the maximum number of rules to print in the rules ODS table.

» **RULESTOPRINT** = *number* «

If this option is not specified, the default is 25.

RULESTOPRINT must be greater than or equal to 1. A decimal value is truncated.

SAMPLEPROBABILITY

Specifies the probability that each item in the input dataset is included in a subset used for data mining. If the **INEST** option is present, this option has no effect.

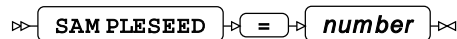
» **SAMPLEPROBABILITY** = *number* «

If this option is not specified, the default is 1.0, which mines the entire input dataset.

Must be between 0 and 1.

SAMPLESEED

Specifies the seed used to initialize the random number generator that selects observations from the input dataset.



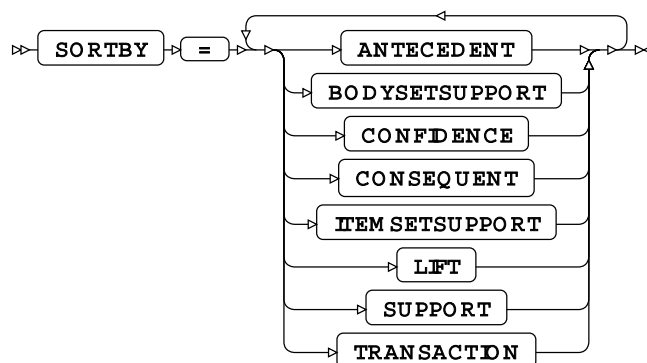
If the `INEST` option is specified, `SAMPLESEED` has no effect.

If this option is not specified, the random number generator is seeded with a value from the system clock.

Must be greater than 0 and less than 1.

SORTBY

Specifies how to sort the association rules or matches in the rules table written to ODS output and the output dataset specified by the `OUTPUT` statement.



If the `INEST` option is not specified, `SORTBY` specifies how to sort the association rules in the rules ODS output and in the output dataset specified by the `OUTPUT` statement.

If the `INEST` option is specified, the `SORTBY` option specifies the fields that will be used to sort the matches generated.

The options specify which column to sort by.

`ANTECEDENT` and `TRANSACTION` sort in lexicographic or numeric ascending order; whereas all other options sort in descending order.

`ANTECEDENT` is only supported when the `INEST` option is not specified.

`TRANSACTION` is only supported when the `INEST` option is specified.

OUTPUT

Describes the options for the `OUTPUT` statement available with the `ASSOCRULES` procedure.

➤ `OUTPUT` ➤ *option* ➤ `;` ➤

Options

The following *options* are available:

OUT

Specifies a WPS dataset that is populated with a human-readable representation of the association rules found. The order of the rules in the dataset can be controlled using the `SORTBY` option.

➤ `OUT` ➤ `=` ➤ *output-dataset* ➤

If the `INEST` option is specified, the `OUTPUT` statement is mandatory and specifies a WPS dataset that is populated with the matches found by matching the rules in the `INEST` dataset with the transactions in the `DATA` dataset. The order of the matches in the dataset can be controlled using the `SORTBY` option.

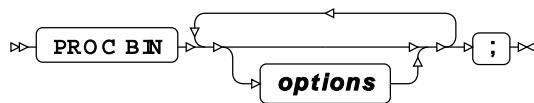
If `INEST` is not specified, the `OUTPUT` statement specifies a WPS dataset that will be populated with the association rules that are found. The order of the rules in the dataset can be controlled using the `SORTBY=` option.

BIN procedure

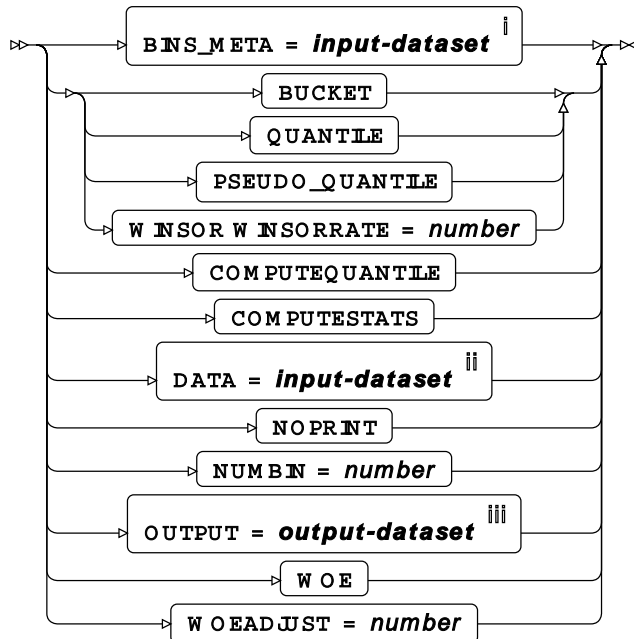
Supported statements

- `PROC BIN` [↗](#) (page 2810)
- `ATTRIB` [↗](#) (page 2810)
- `BY` [↗](#) (page 2811)
- `FORMAT` [↗](#) (page 2811)
- `FREQ` [↗](#) (page 2811)
- `INFORMAT` [↗](#) (page 2811)
- `LABEL` [↗](#) (page 2811)
- `ID` [↗](#) (page 2812)
- `INPUT` [↗](#) (page 2812)
- `TARGET` [↗](#) (page 2812)
- `WHERE` [↗](#) (page 2812)

PROC BIN



options

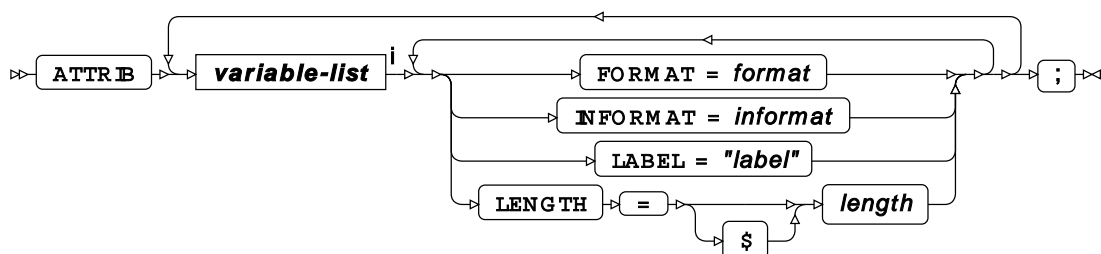


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

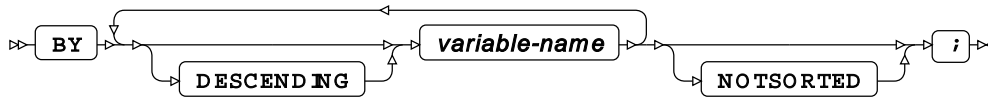
ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

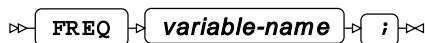


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

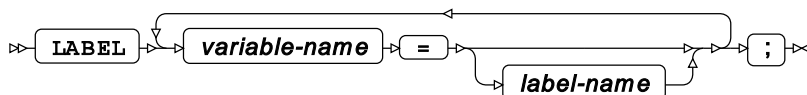


INFORMAT

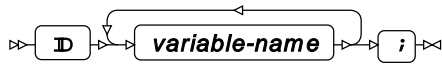


ⁱ See *Variable Lists* [↗](#) (page 32).

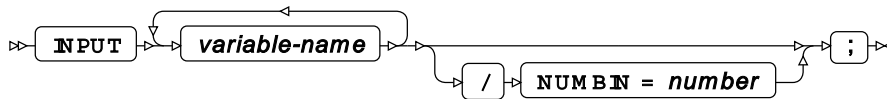
LABEL



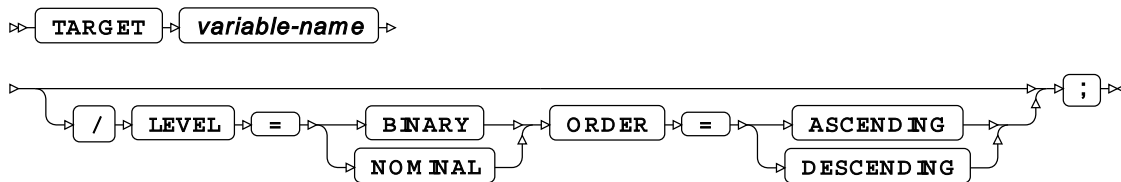
ID



INPUT



TARGET



WHERE



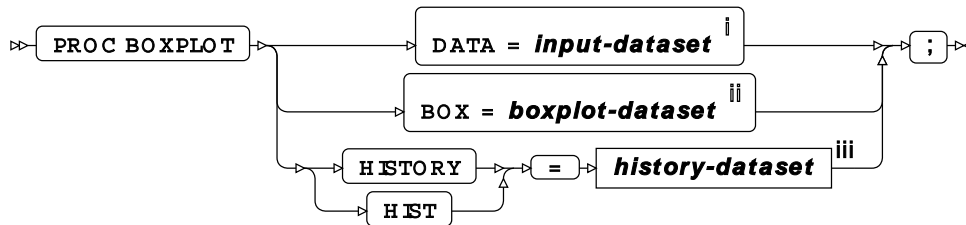
BOXPLOT procedure

Supported statements

- *PROC* **BOXPLOT** [↗](#) (page 2813)
- *ATTRIB* [↗](#) (page 2813)
- *BY* [↗](#) (page 2813)
- *FORMAT* [↗](#) (page 2814)
- *ID* [↗](#) (page 2814)
- *INFORMAT* [↗](#) (page 2814)
- *LABEL* [↗](#) (page 2814)
- *PLOT* [↗](#) (page 2814)

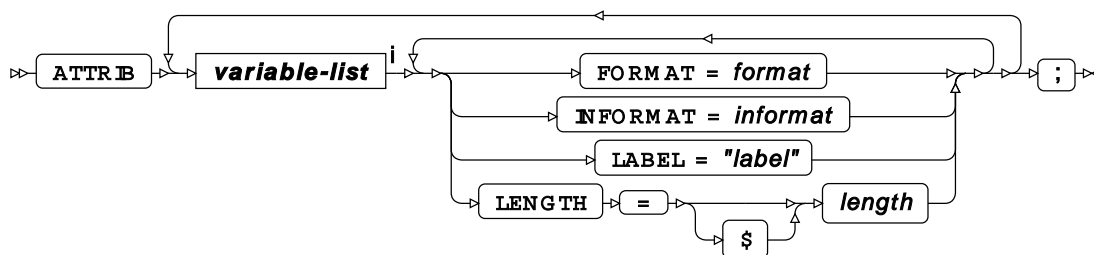
- [WHERE](#) (page 2816)

PROC BOXPLOT



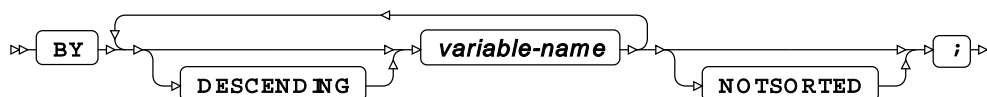
- ⁱ See [Input dataset](#) (page 16).
- ⁱⁱ See [Input dataset](#) (page 16).
- ⁱⁱⁱ See [Input dataset](#) (page 16).

ATTRIB

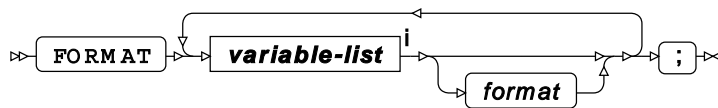


- ⁱ See [Variable Lists](#) (page 32).

BY

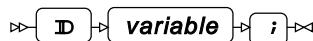


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

ID

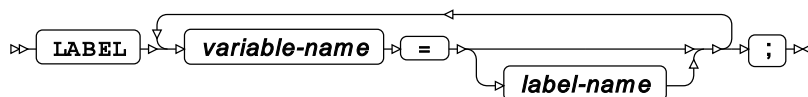


INFORMAT

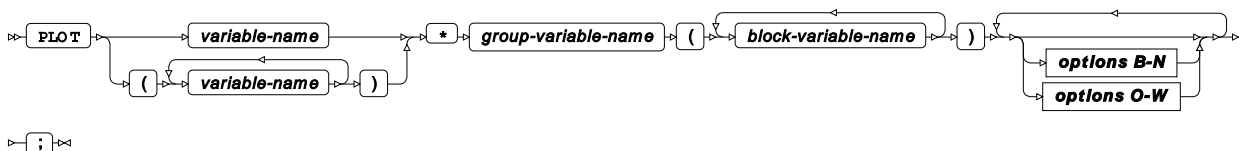


ⁱ See *Variable Lists* [↗](#) (page 32).

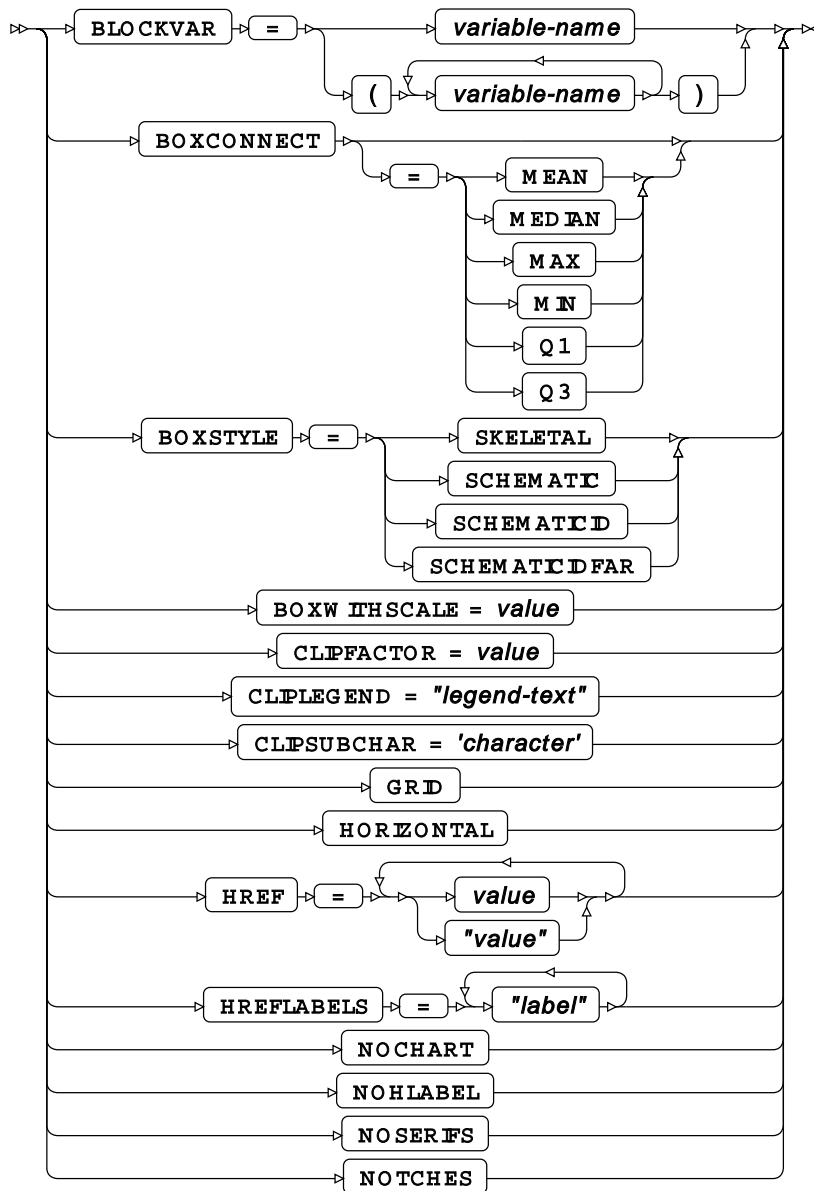
LABEL



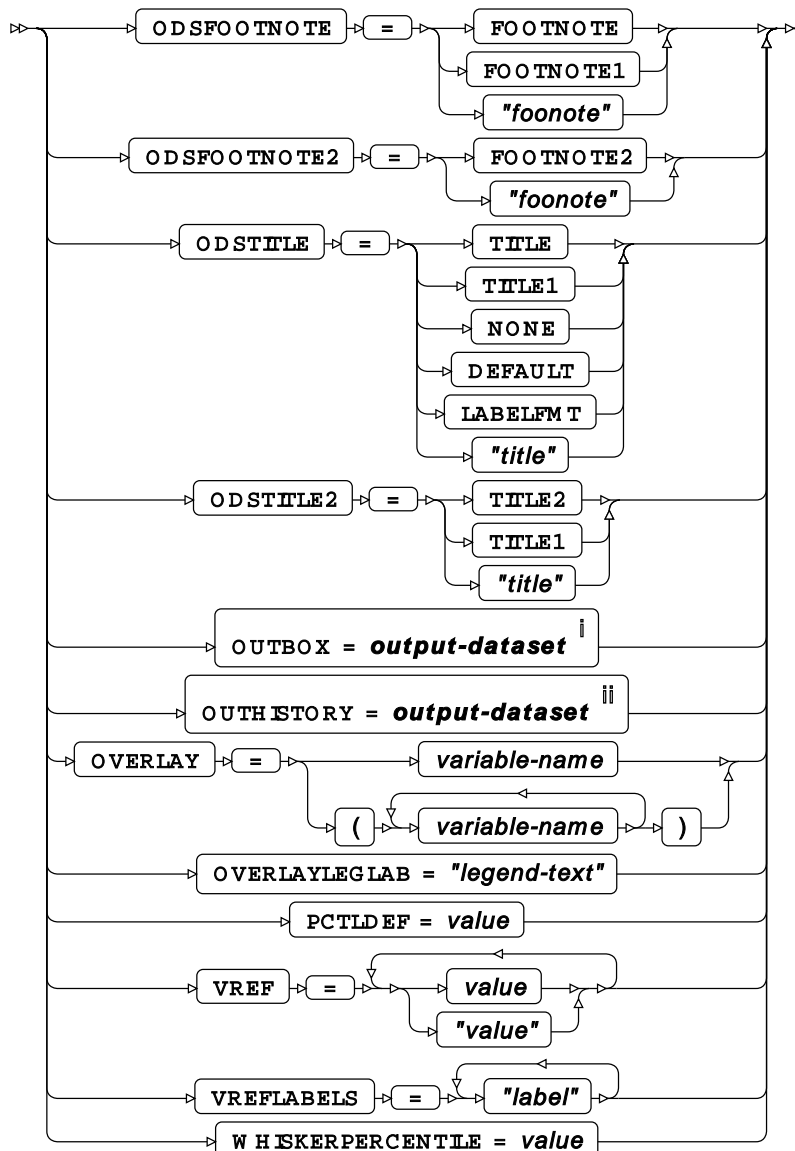
PLOT



options B-N



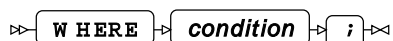
options O-W



ⁱ See *Output dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

WHERE

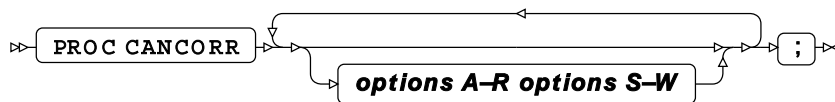


CANCORR procedure

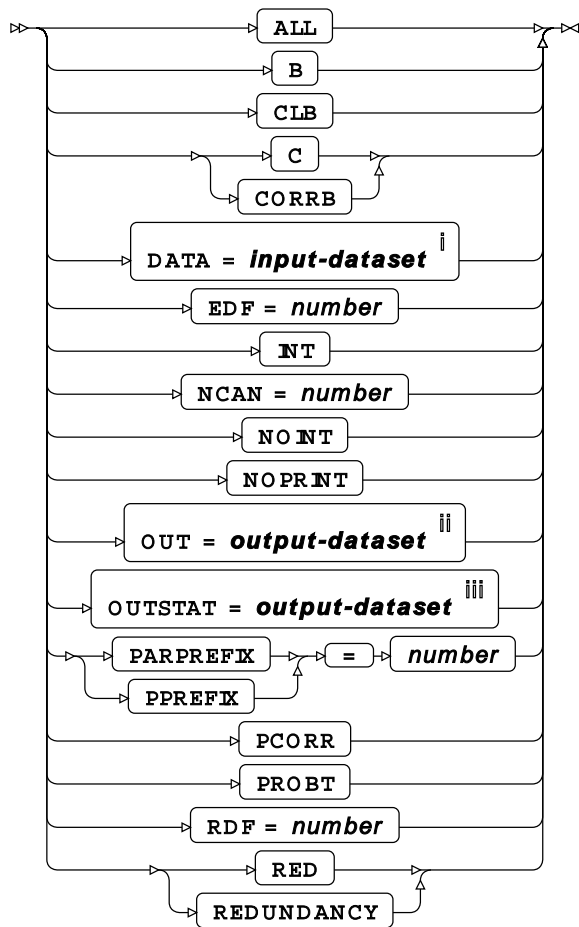
Supported statements

- *PROC CANCORR* [↗](#) (page 2817)
- *ATTRIB* [↗](#) (page 2819)
- *BY* [↗](#) (page 2820)
- *FORMAT* [↗](#) (page 2820)
- *FREQ* [↗](#) (page 2820)
- *INFORMAT* [↗](#) (page 2820)
- *LABEL* [↗](#) (page 2820)
- *PARTIAL* [↗](#) (page 2821)
- *VAR* [↗](#) (page 2821)
- *WEIGHT* [↗](#) (page 2821)
- *WHERE* [↗](#) (page 2821)
- *WITH* [↗](#) (page 2821)

PROC CANCORR



options A–R

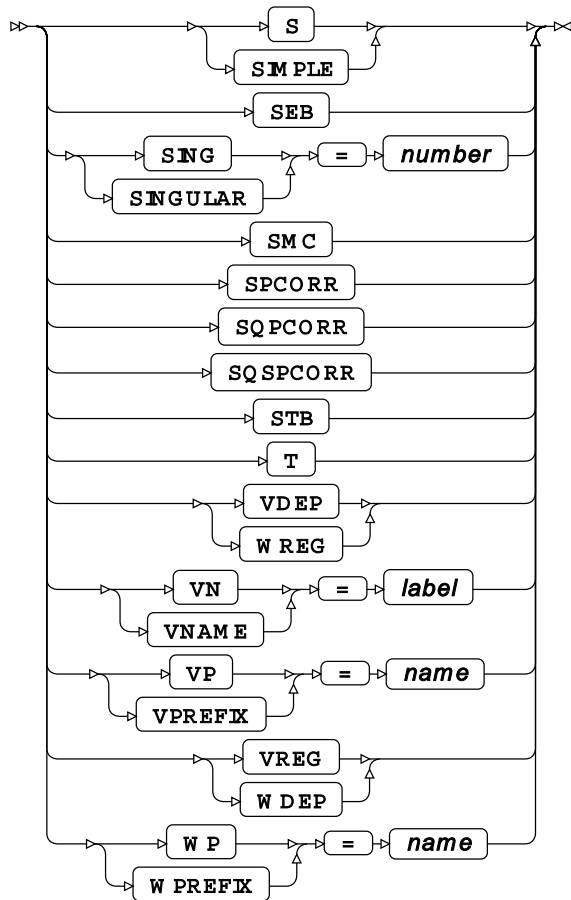


ⁱ See *Input dataset* [↗](#) (page 16).

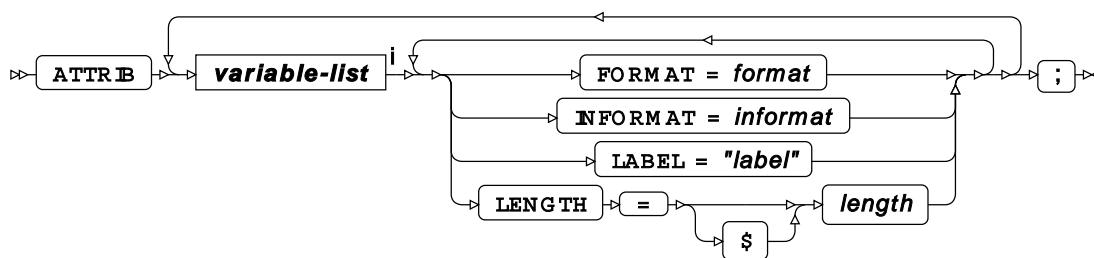
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

options S–W

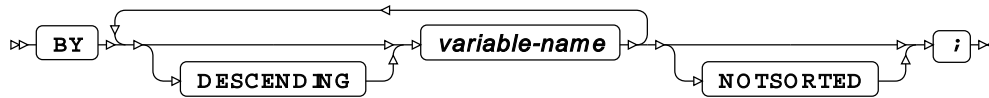


ATTRIB

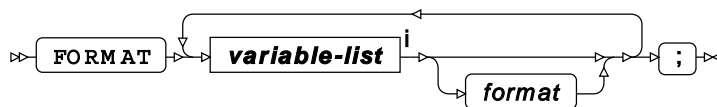


ⁱ See [Variable Lists](#) (page 32).

BY

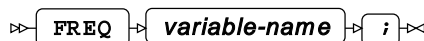


FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

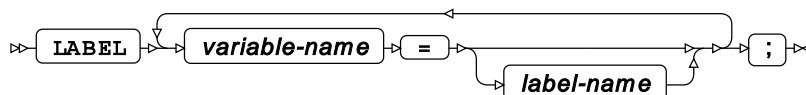


INFORMAT

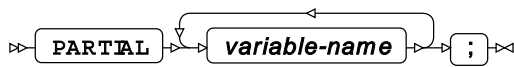


ⁱ See [Variable Lists](#) (page 32).

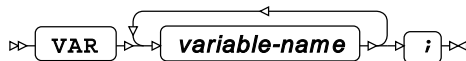
LABEL



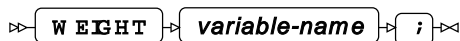
PARTIAL



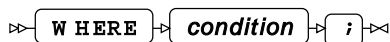
VAR



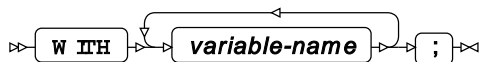
WEIGHT



WHERE



WITH



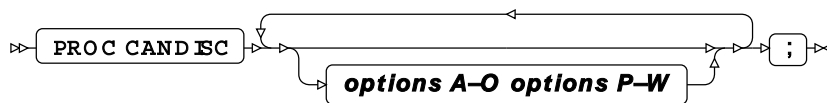
CANDISC procedure

Supported statements

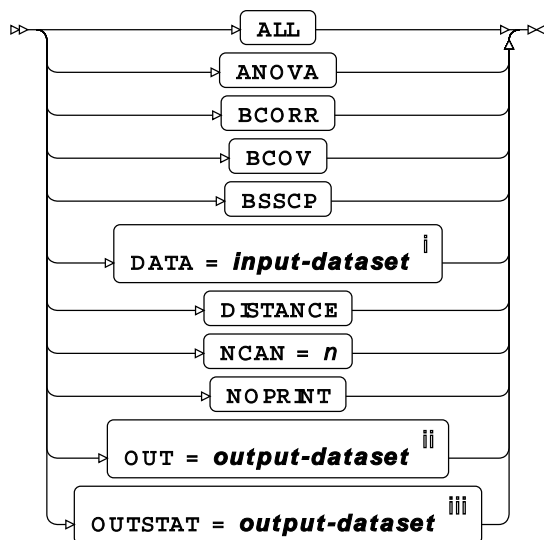
- *PROC CANDISC* [↗](#) (page 2822)
- *ATTRIB* [↗](#) (page 2823)
- *BY* [↗](#) (page 2823)
- *CLASS* [↗](#) (page 2824)
- *FORMAT* [↗](#) (page 2824)
- *FREQ* [↗](#) (page 2824)

- [INFORMAT](#) (page 2824)
- [LABEL](#) (page 2824)
- [VAR](#) (page 2825)
- [WEIGHT](#) (page 2825)
- [WHERE](#) (page 2825)

PROC CANDISC



options A–O

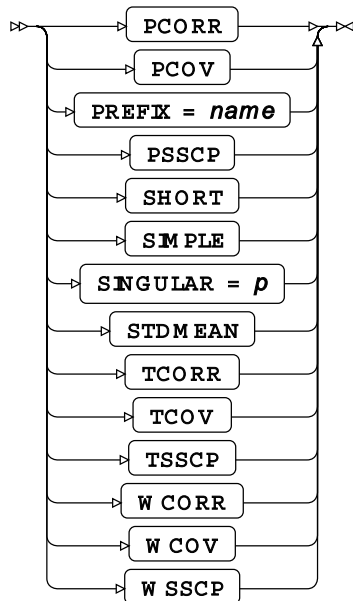


ⁱ See [Input dataset](#) (page 16).

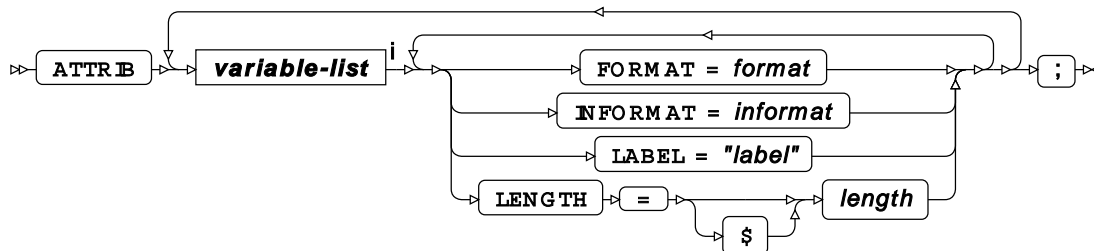
ⁱⁱ See [Input dataset](#) (page 16).

ⁱⁱⁱ See [Input dataset](#) (page 16).

options P–W

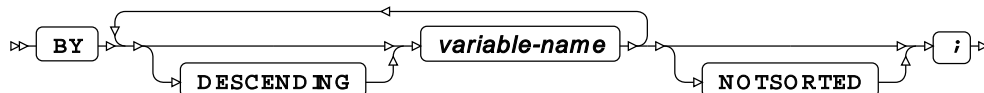


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

BY



CLASS

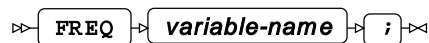


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

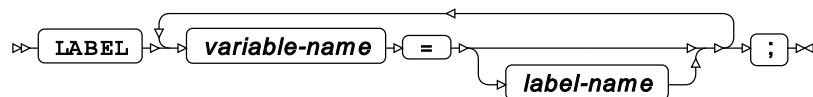


INFORMAT

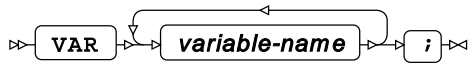


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



VAR



WEIGHT



WHERE

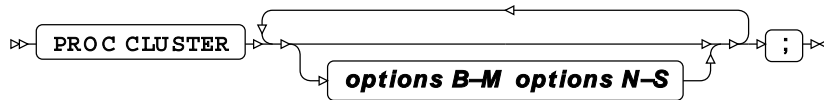


CLUSTER procedure

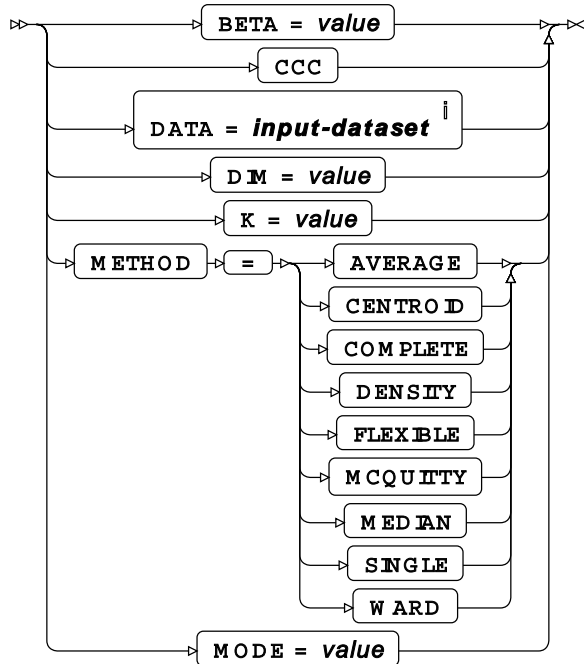
Supported statements

- *PROC CLUSTER* [↗](#) (page 2826)
- *ATTRIB* [↗](#) (page 2827)
- *BY* [↗](#) (page 2827)
- *COPY* [↗](#) (page 2828)
- *FORMAT* [↗](#) (page 2828)
- *FREQ* [↗](#) (page 2828)
- *ID* [↗](#) (page 2828)
- *INFORMAT* [↗](#) (page 2828)
- *LABEL* [↗](#) (page 2828)
- *RMSSTD* [↗](#) (page 2829)
- *VAR* [↗](#) (page 2829)
- *WHERE* [↗](#) (page 2829)

PROC CLUSTER

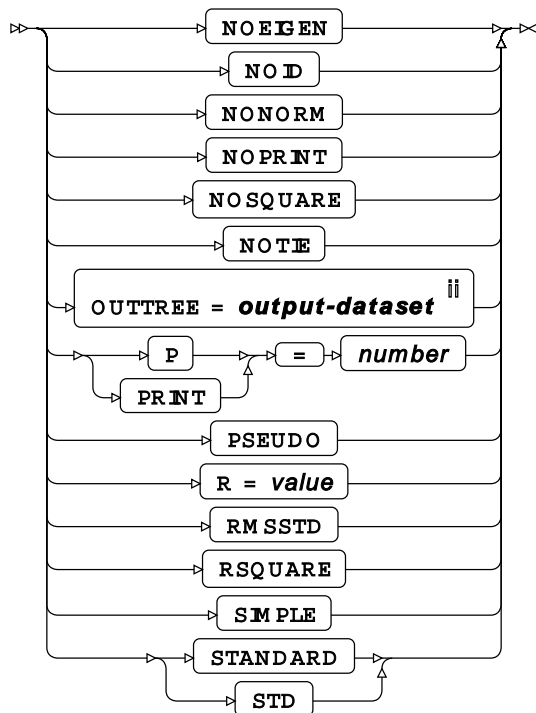


options B–M



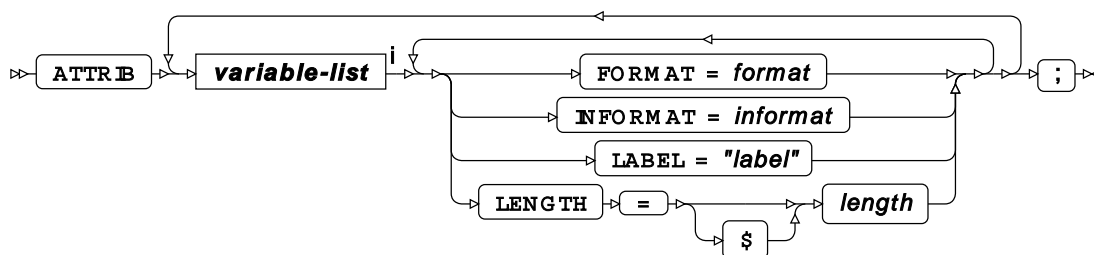
ⁱ See [Input dataset](#) (page 16).

options N–S



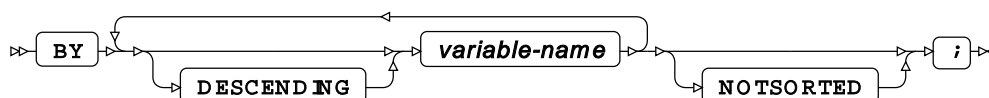
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

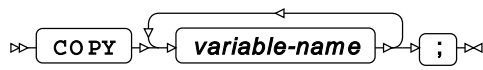


ⁱ See *Variable Lists* [↗](#) (page 32).

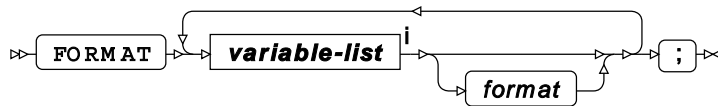
BY



COPY

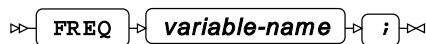


FORMAT

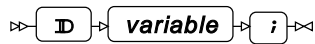


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

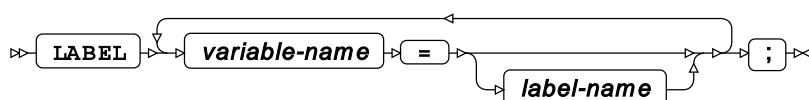


INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

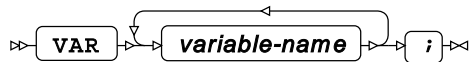
LABEL



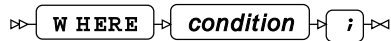
RMSSTD



VAR



WHERE

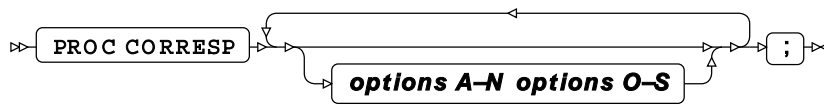


CORRESP procedure

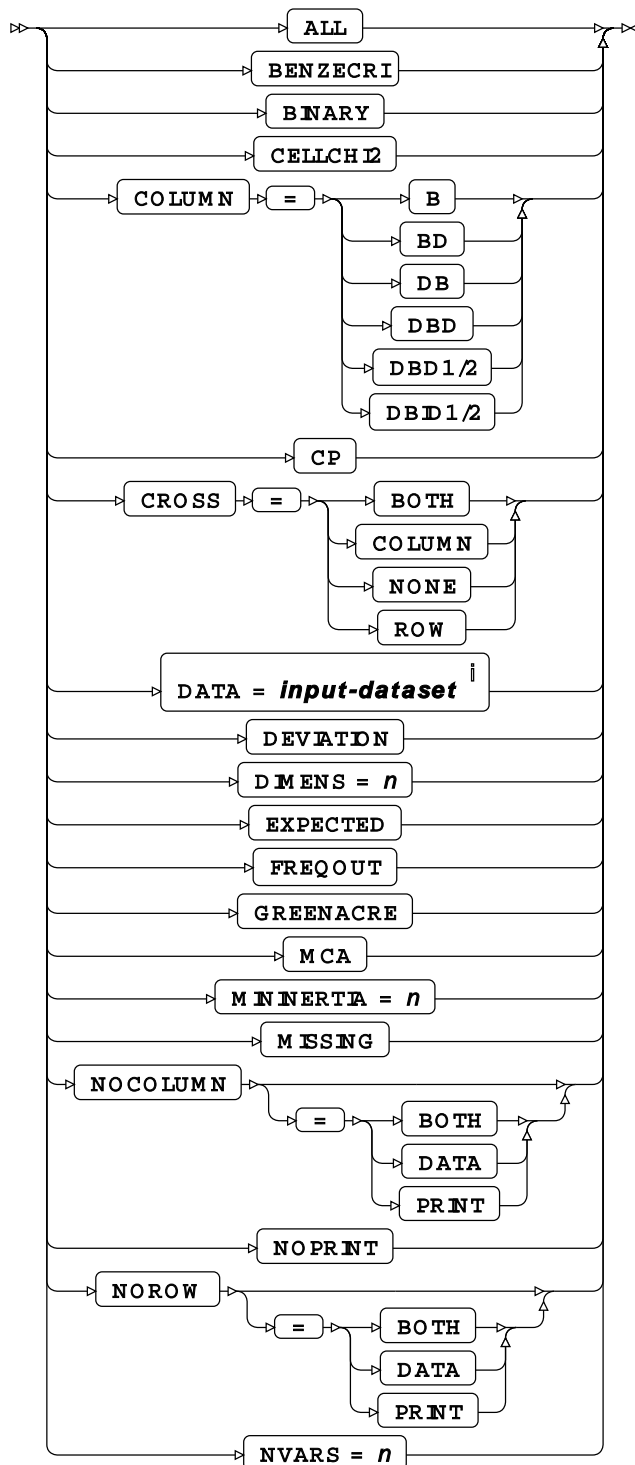
Supported statements

- *PROC CORRESP* [↗](#) (page 2830)
- *ATTRIB* [↗](#) (page 2833)
- *BY* [↗](#) (page 2833)
- *FORMAT* [↗](#) (page 2833)
- *ID* [↗](#) (page 2833)
- *INFORMAT* [↗](#) (page 2833)
- *LABEL* [↗](#) (page 2834)
- *SUPPLEMENTARY* [↗](#) (page 2834)
- *TABLES* [↗](#) (page 2834)
- *VAR* [↗](#) (page 2834)
- *WEIGHT* [↗](#) (page 2834)
- *WHERE* [↗](#) (page 2834)

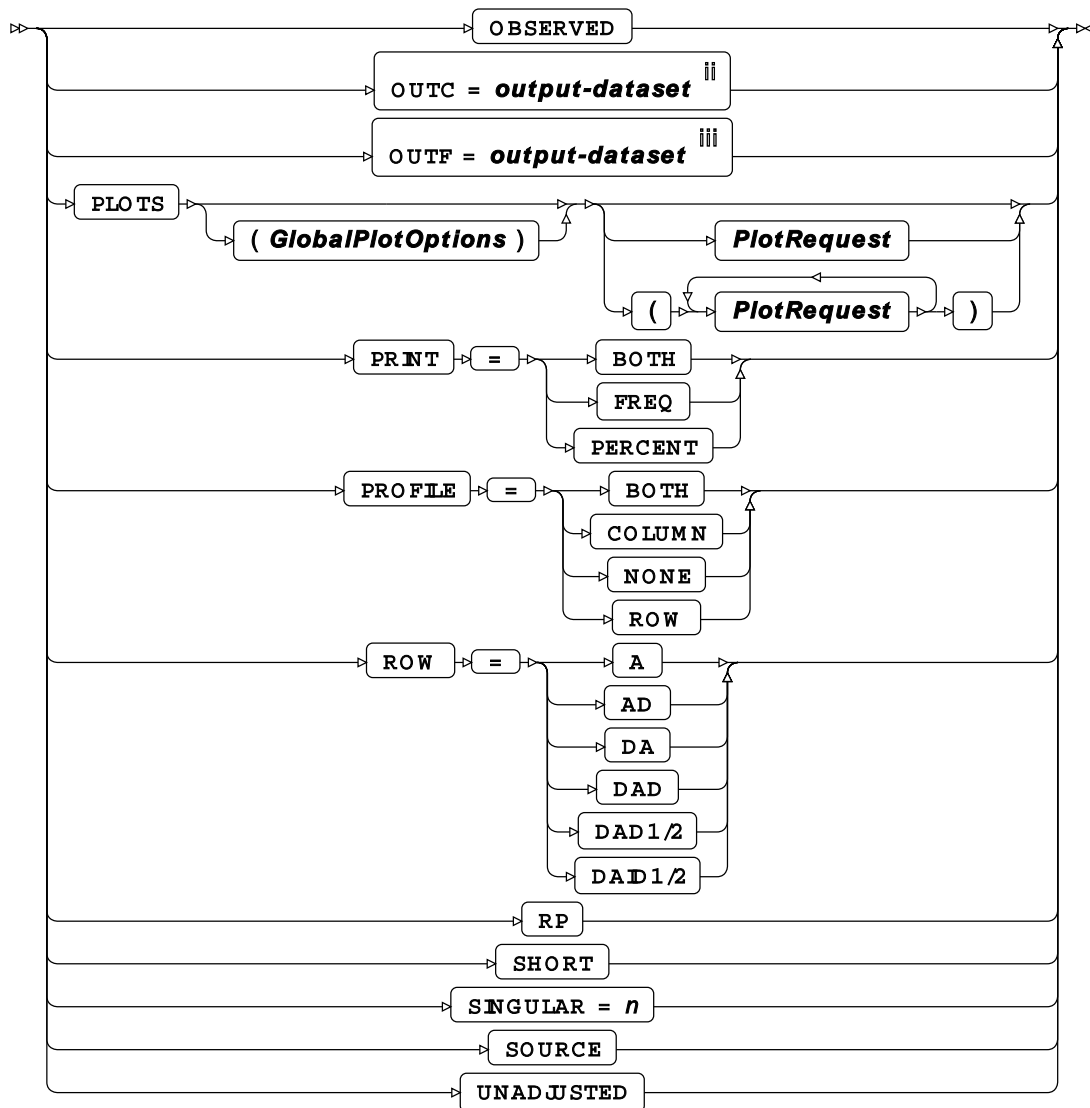
PROC CORRESP



options A–N

ⁱ See *Input dataset* [↗](#) (page 16).

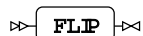
options O–S



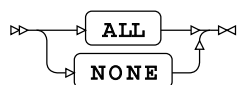
ii See [Input dataset](#) (page 16).

iii See [Input dataset](#) (page 16).

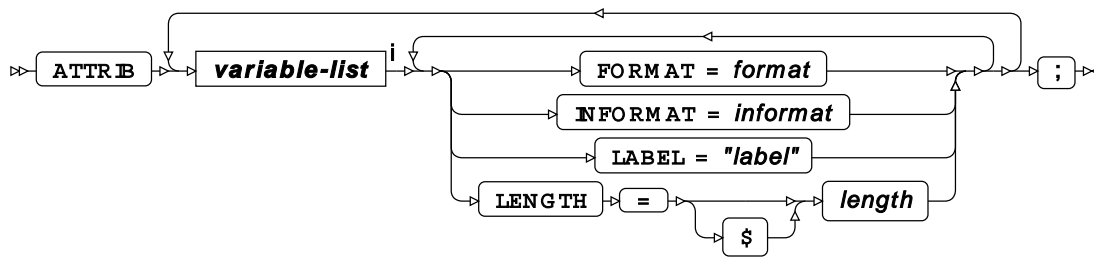
GlobalPlotOptions



PlotRequest

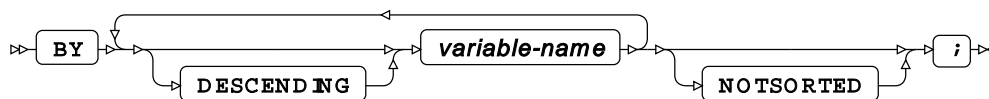


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

ID

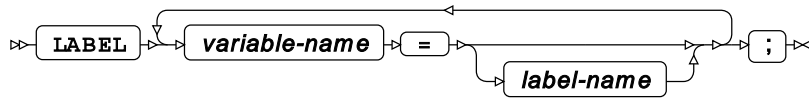


INFORMAT

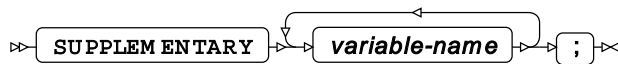


ⁱ See *Variable Lists* [↗](#) (page 32).

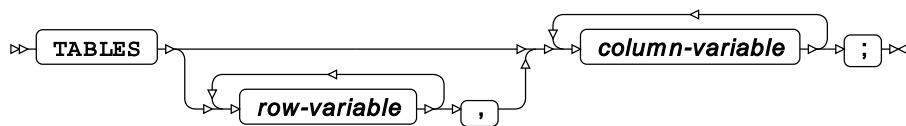
LABEL



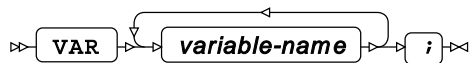
SUPPLEMENTARY



TABLES



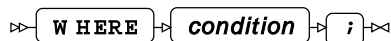
VAR



WEIGHT



WHERE

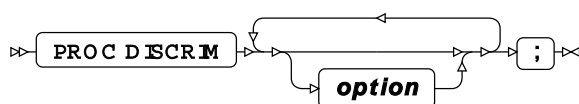


DISCRIM procedure

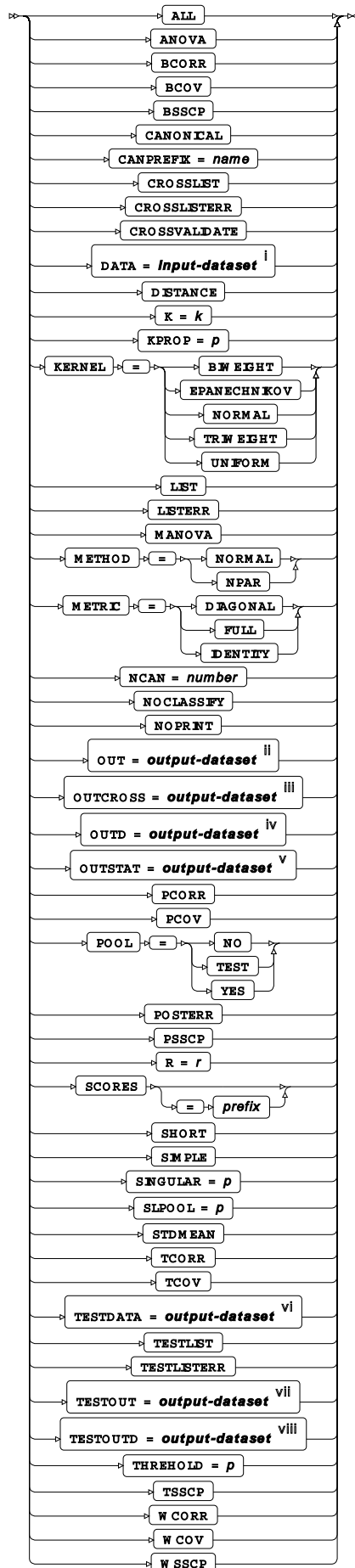
Supported statements

- *PROC DISCRIM* [↗](#) (page 2835)
- *ATTRIB* [↗](#) (page 2837)
- *BY* [↗](#) (page 2837)
- *CLASS* [↗](#) (page 2837)
- *FORMAT* [↗](#) (page 2838)
- *FREQ* [↗](#) (page 2838)
- *ID* [↗](#) (page 2838)
- *INFORMAT* [↗](#) (page 2838)
- *LABEL* [↗](#) (page 2838)
- *PRIORS* [↗](#) (page 2839)
- *TESTCLASS* [↗](#) (page 2839)
- *TESTFREQ* [↗](#) (page 2839)
- *TESTID* [↗](#) (page 2839)
- *VAR* [↗](#) (page 2839)
- *WEIGHT* [↗](#) (page 2839)
- *WHERE* [↗](#) (page 2839)

PROC DISCRIM

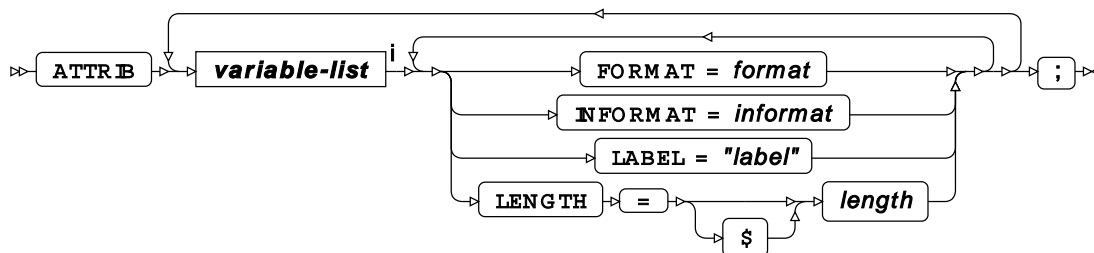


option



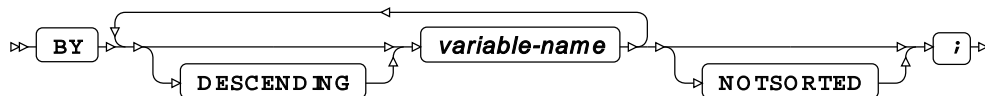
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).
- ^{iv} See *Input dataset* [↗](#) (page 16).
- ^v See *Input dataset* [↗](#) (page 16).
- ^{vi} See *Input dataset* [↗](#) (page 16).
- ^{vii} See *Input dataset* [↗](#) (page 16).
- ^{viii} See *Input dataset* [↗](#) (page 16).

ATTRIB



- ⁱ See *Variable Lists* [↗](#) (page 32).

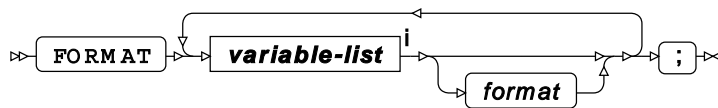
BY



CLASS

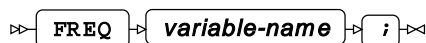


FORMAT

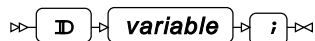


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

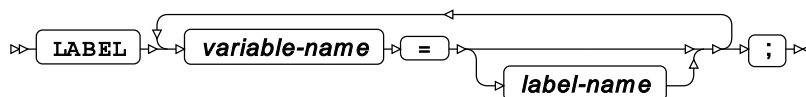


INFORMAT

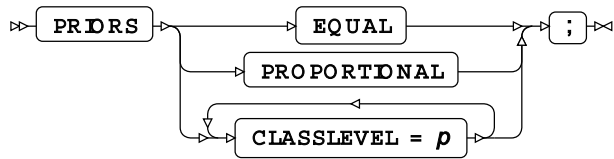


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



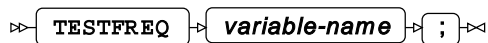
PRIORS



TESTCLASS



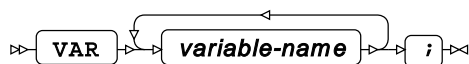
TESTFREQ



TESTID



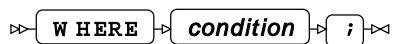
VAR



WEIGHT



WHERE

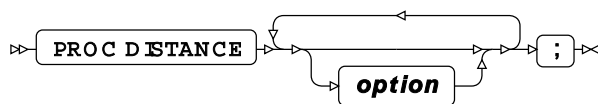


DISTANCE procedure

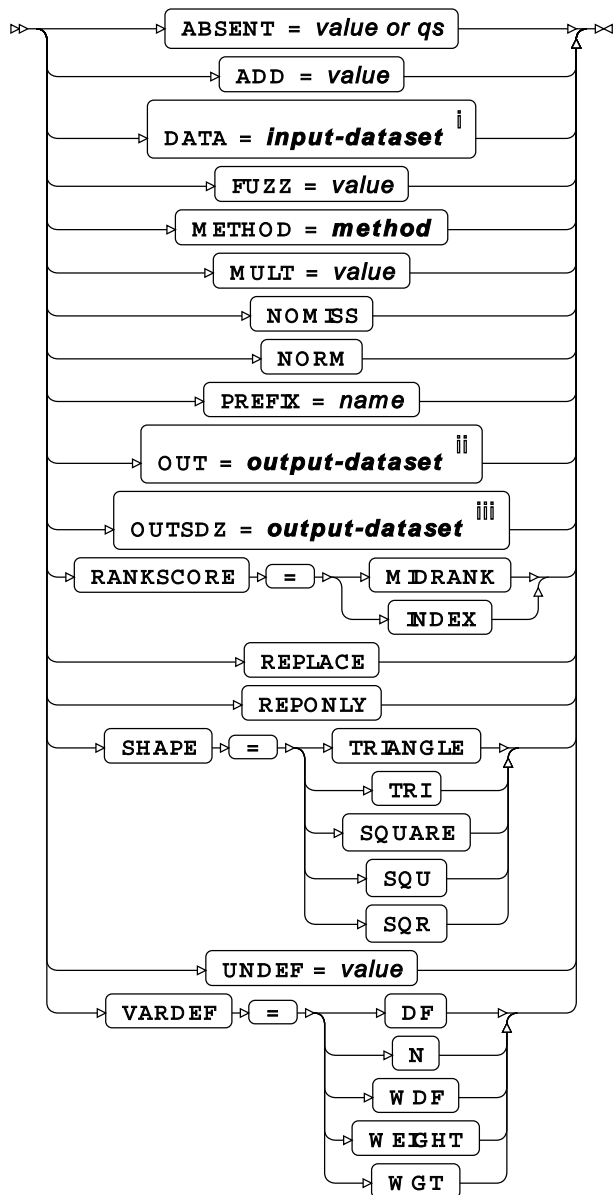
Supported statements

- *PROC DISTANCE* [↗](#) (page 2840)
- *ATTRIB* [↗](#) (page 2843)
- *BY* [↗](#) (page 2843)
- *COPY* [↗](#) (page 2843)
- *FORMAT* [↗](#) (page 2843)
- *FREQ* [↗](#) (page 2843)
- *ID* [↗](#) (page 2844)
- *INFORMAT* [↗](#) (page 2844)
- *LABEL* [↗](#) (page 2844)
- *VAR* [↗](#) (page 2844)
- *WEIGHT* [↗](#) (page 2845)
- *WHERE* [↗](#) (page 2845)

PROC DISTANCE



option

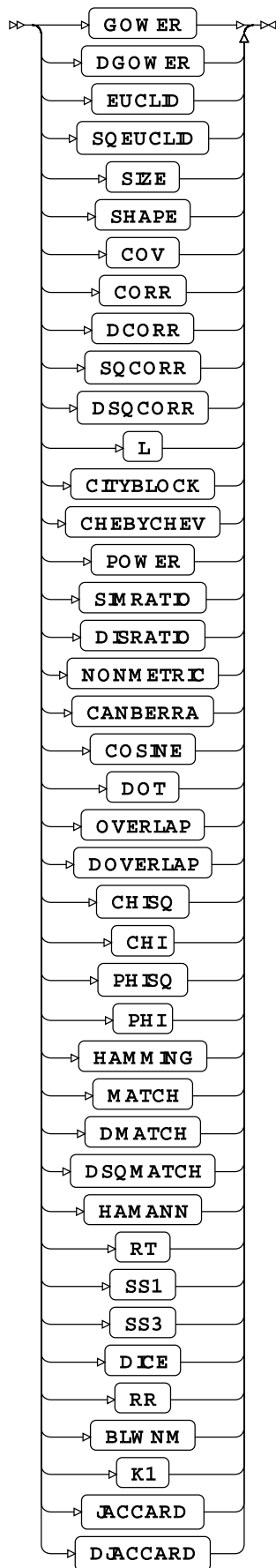


ⁱ See *Input dataset* [↗](#) (page 16).

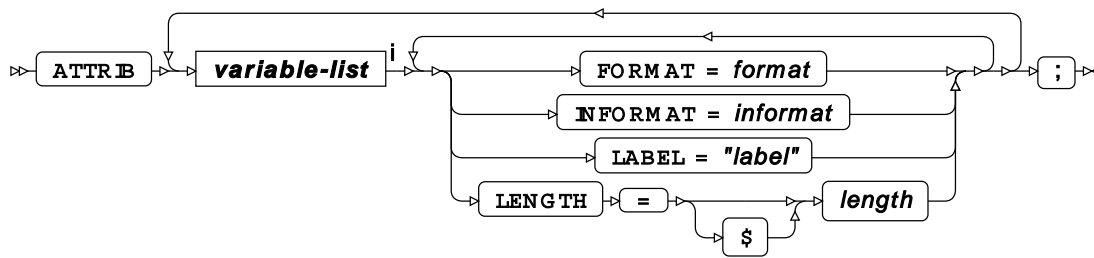
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

method

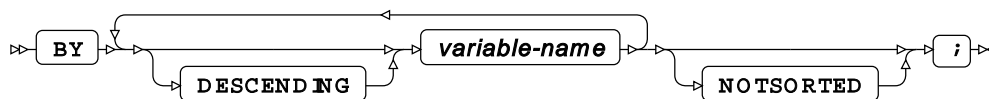


ATTRIB

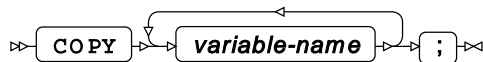


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



COPY

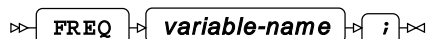


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

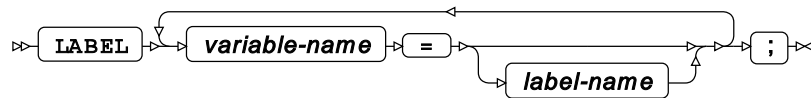


INFORMAT

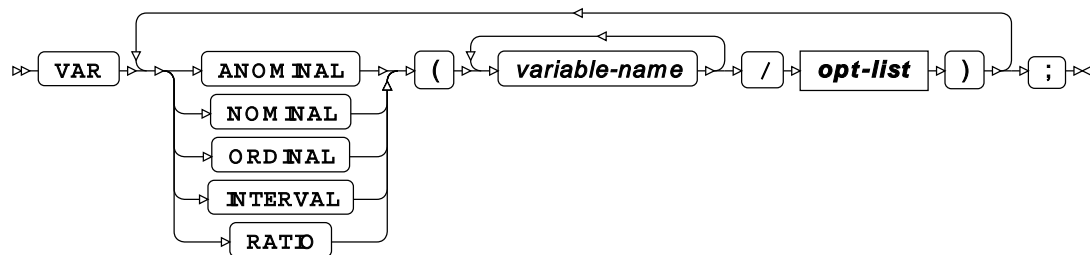


ⁱ See [Variable Lists](#) (page 32).

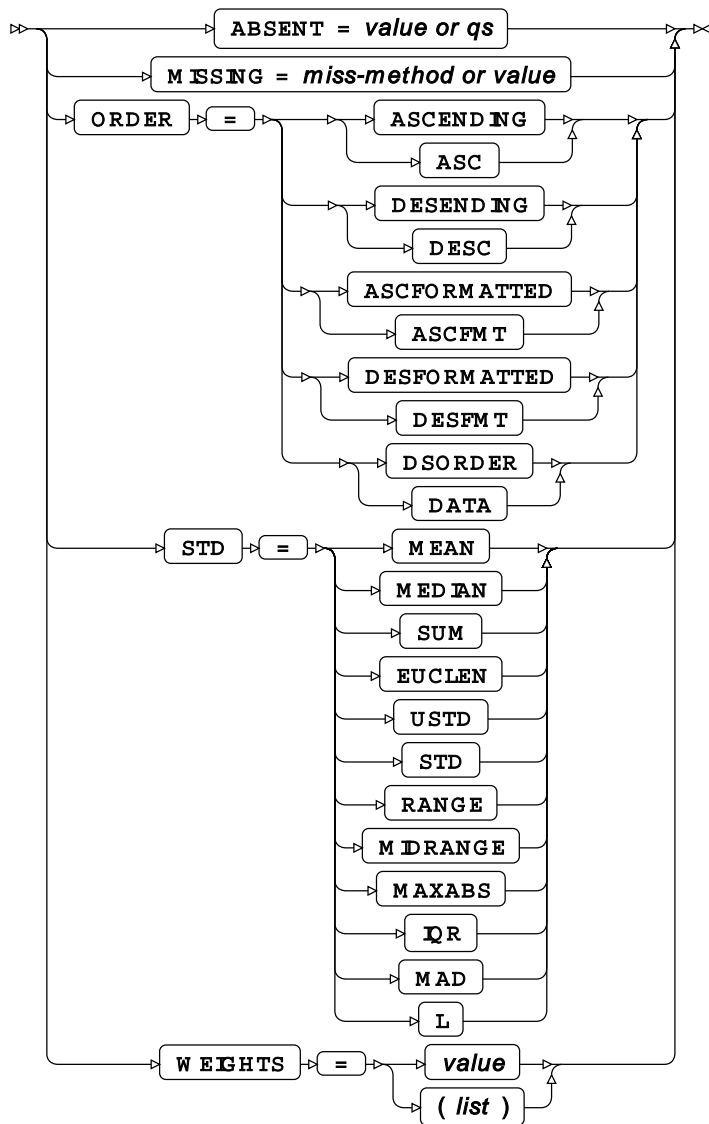
LABEL



VAR



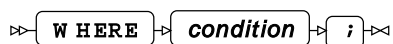
opt-list



WEIGHT



WHERE

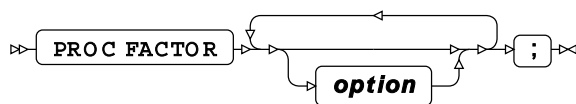


FACTOR procedure

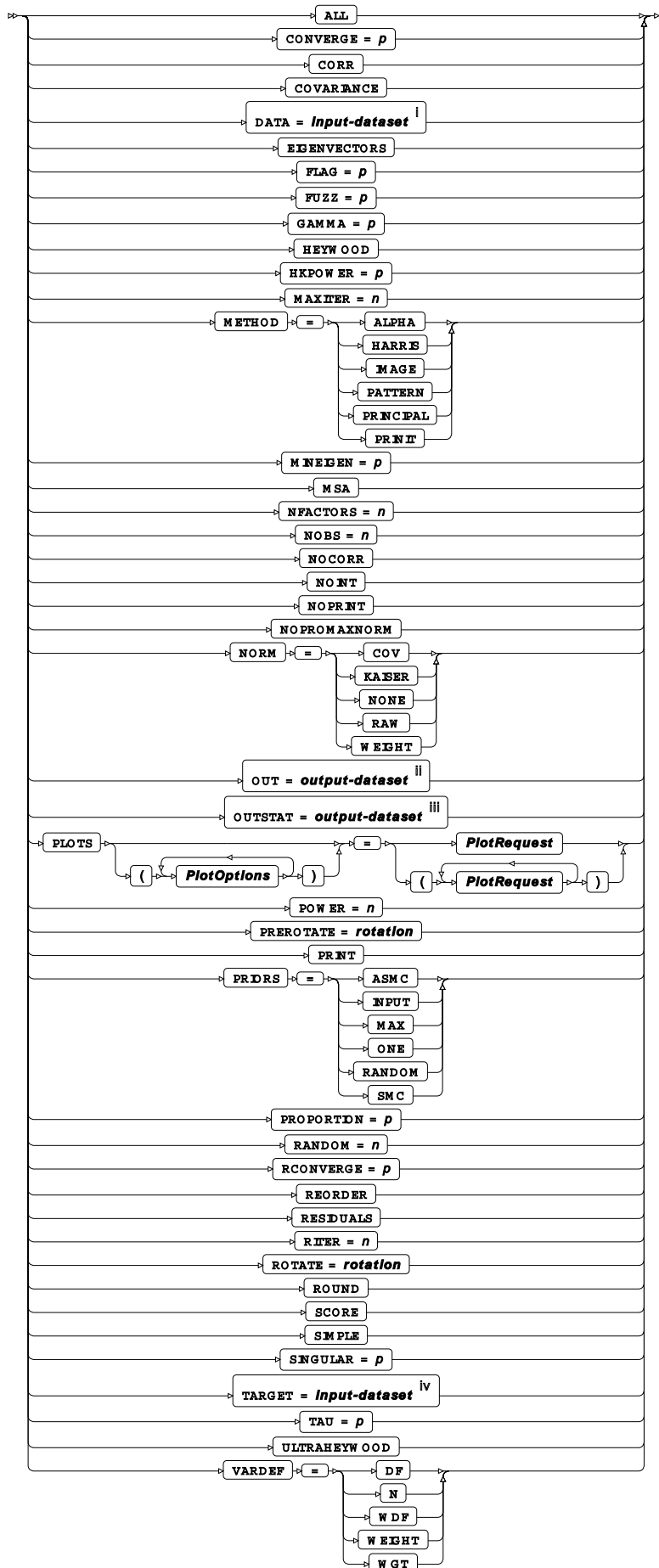
Supported statements

- *PROC FACTOR* [↗](#) (page 2846)
- *ATTRIB* [↗](#) (page 2849)
- *BY* [↗](#) (page 2850)
- *FORMAT* [↗](#) (page 2850)
- *FREQ* [↗](#) (page 2850)
- *INFORMAT* [↗](#) (page 2850)
- *LABEL* [↗](#) (page 2850)
- *PRIORS* [↗](#) (page 2851)
- *VAR* [↗](#) (page 2851)
- *PARTIAL* [↗](#) (page 2851)
- *WEIGHT* [↗](#) (page 2851)
- *WHERE* [↗](#) (page 2851)

PROC FACTOR

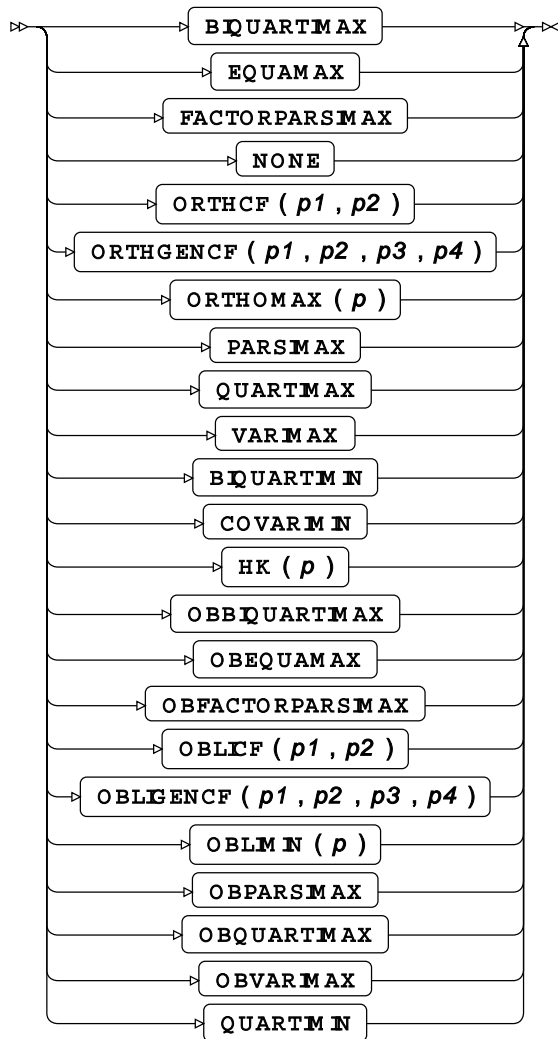


option

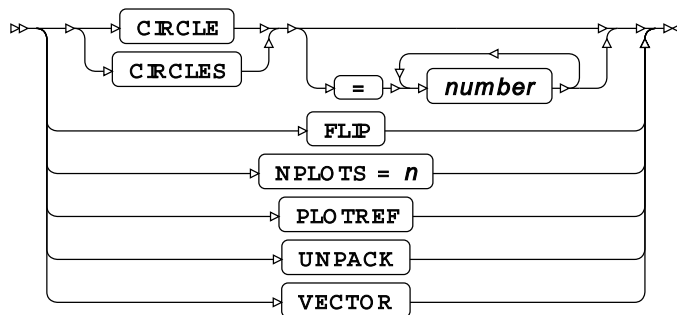


- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).
- ^{iv} See *Input dataset* [↗](#) (page 16).

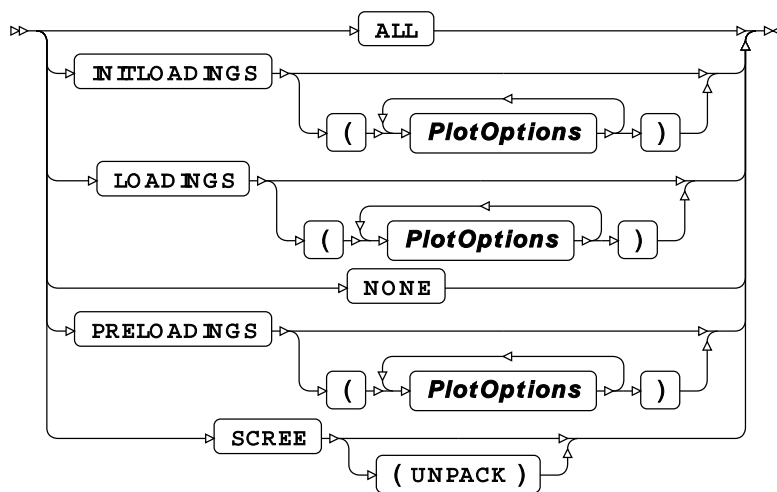
rotation



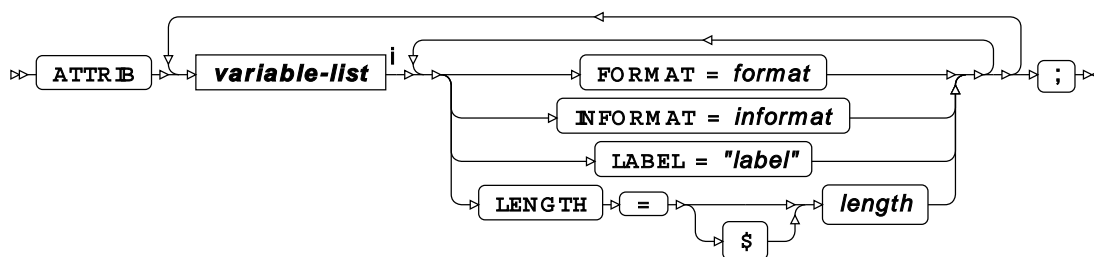
PlotOptions



PlotRequest

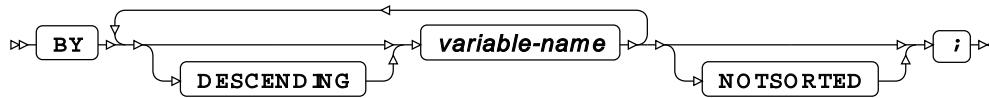


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

BY

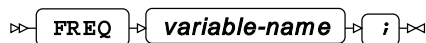


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

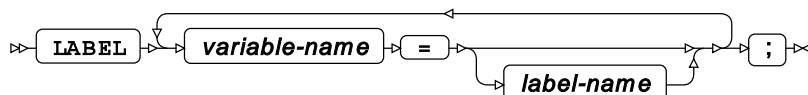


INFORMAT

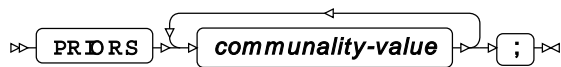


ⁱ See *Variable Lists* [↗](#) (page 32).

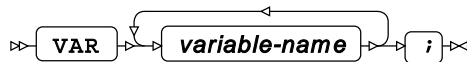
LABEL



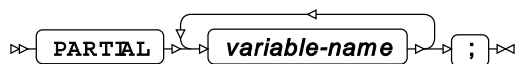
PRIORS



VAR



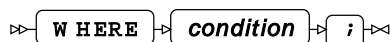
PARTIAL



WEIGHT



WHERE



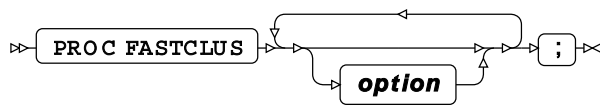
FASTCLUS procedure

Supported statements

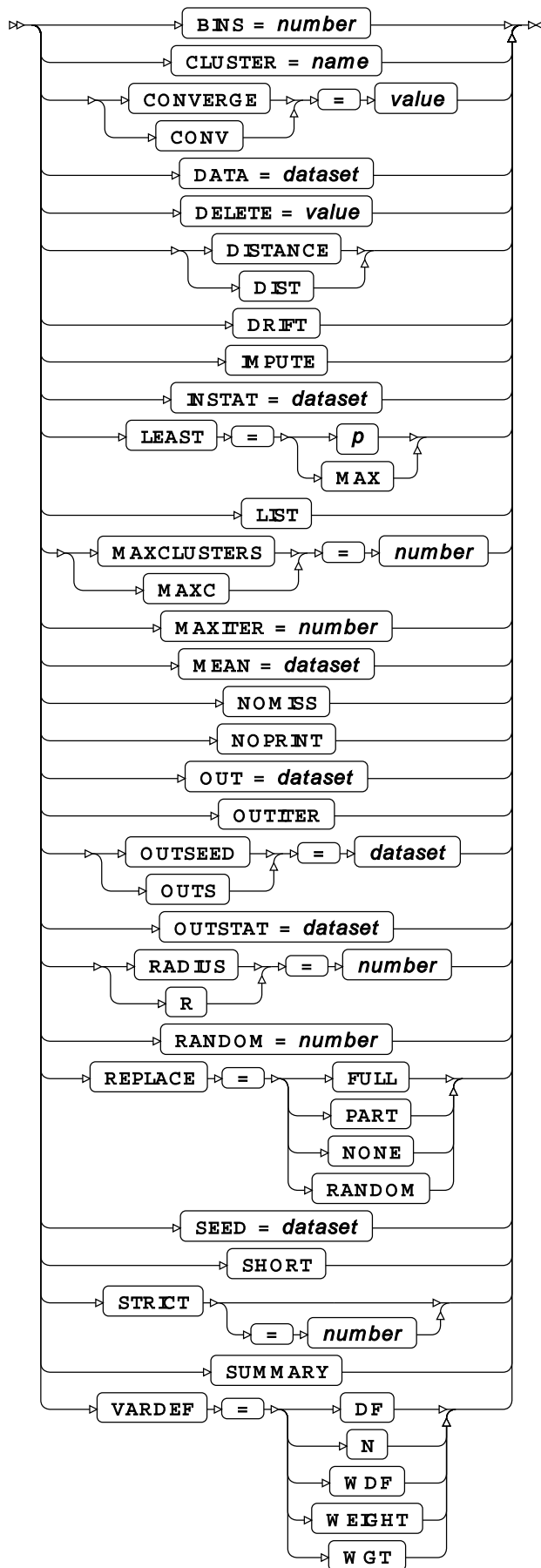
- **PROC FASTCLUS** [↗](#) (page 2852)
- **ATTRIB** [↗](#) (page 2854)
- **BY** [↗](#) (page 2854)
- **FORMAT** [↗](#) (page 2854)
- **FREQ** [↗](#) (page 2854)
- **ID** [↗](#) (page 2854)

- *INFORMAT* [↗](#) (page 2855)
- *LABEL* [↗](#) (page 2855)
- *VAR* [↗](#) (page 2855)
- *WEIGHT* [↗](#) (page 2855)
- *WHERE* [↗](#) (page 2855)

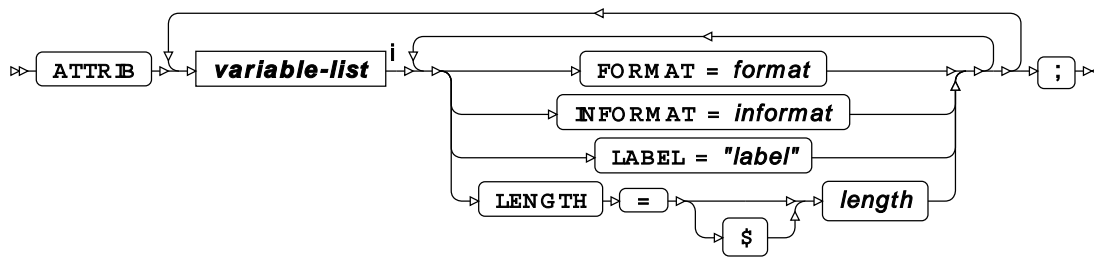
PROC FASTCLUS



option

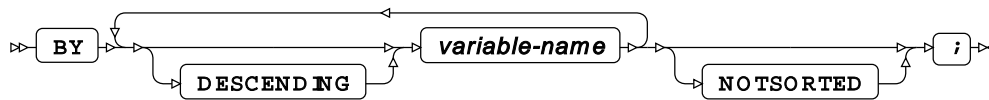


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY



FORMAT

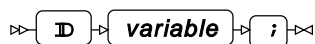


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

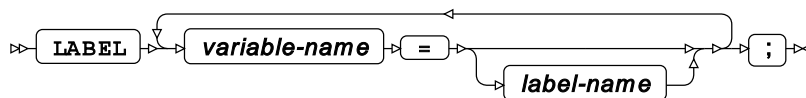


INFORMAT

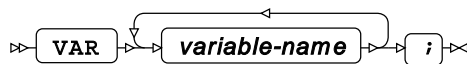


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



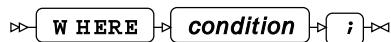
VAR



WEIGHT



WHERE



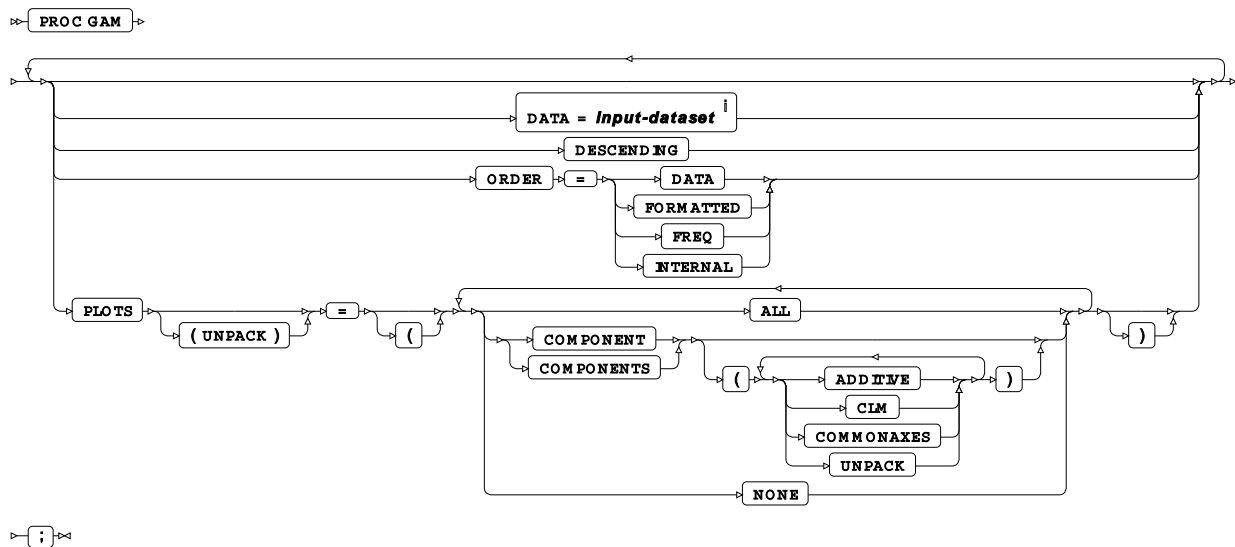
GAM procedure

Supported statements

- *PROC GAM* [↗](#) (page 2856)
- *ATTRIB* [↗](#) (page 2856)
- *BY* [↗](#) (page 2857)

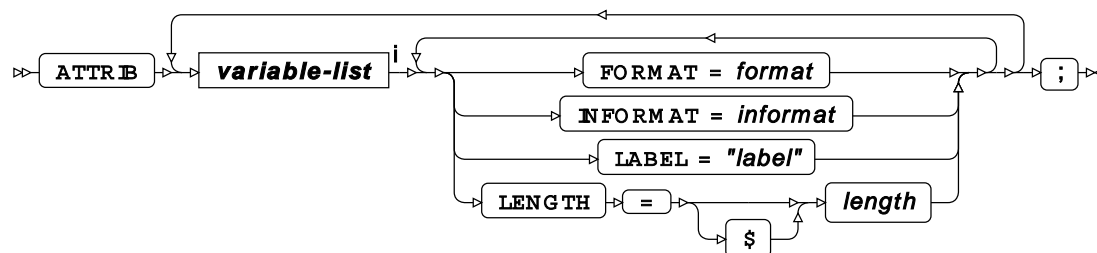
- [CLASS](#) (page 2857)
- [FORMAT](#) (page 2857)
- [FREQ](#) (page 2858)
- [INFORMAT](#) (page 2858)
- [LABEL](#) (page 2858)
- [MODEL](#) (page 2858)
- [OUTPUT](#) (page 2860)
- [SCORE](#) (page 2860)
- [WHERE](#) (page 2860)

PROC GAM



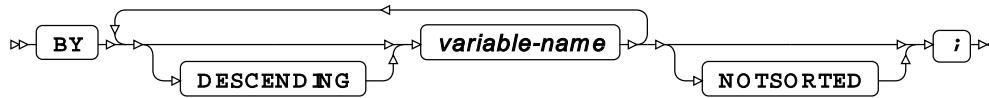
ⁱ See *Input dataset* (page 16).

ATTRIB

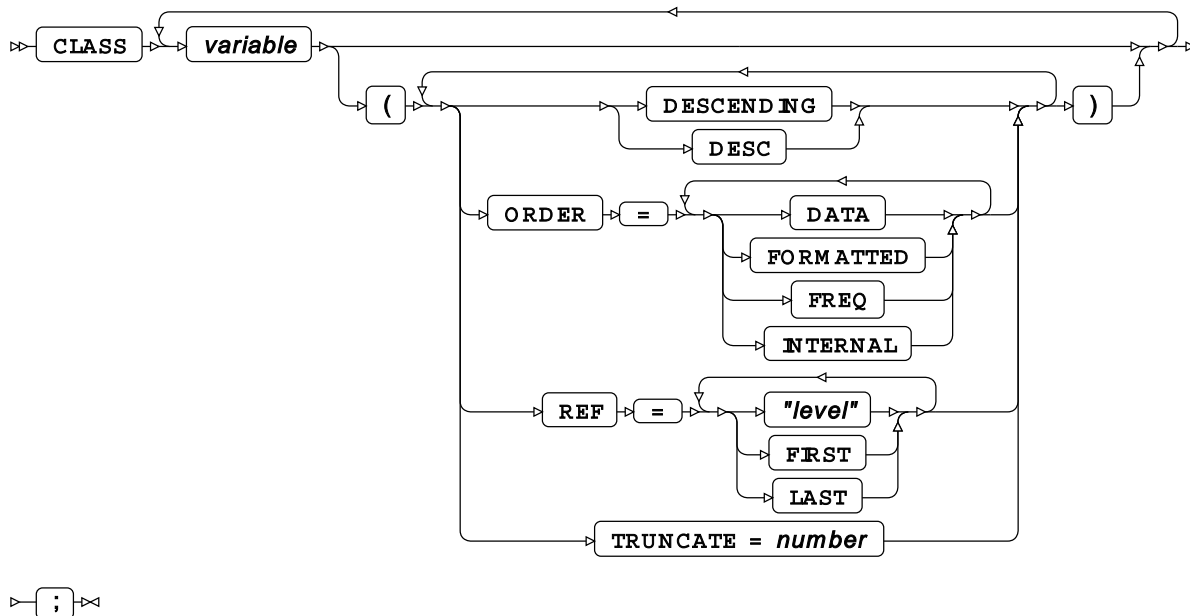


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



CLASS



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

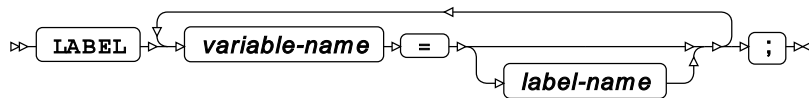


INFORMAT

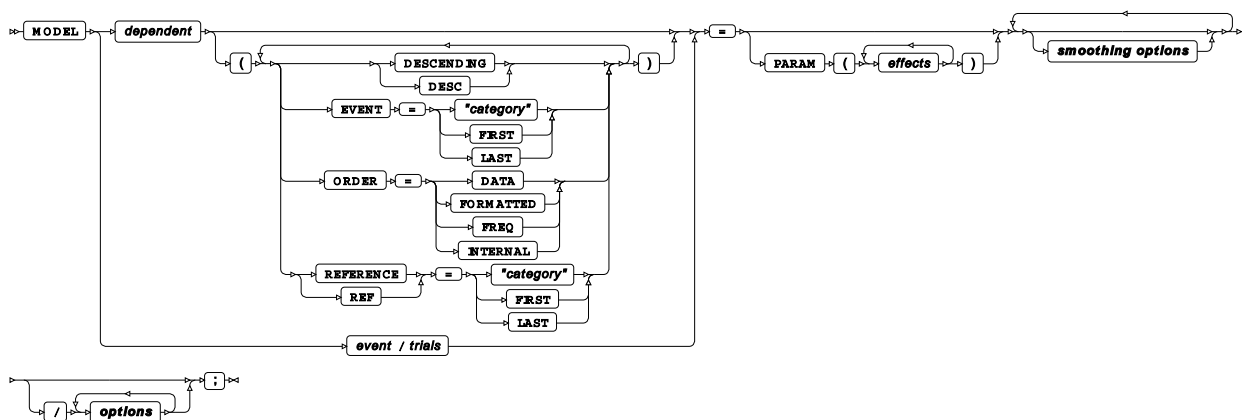


ⁱ See *Variable Lists* [↗](#) (page 32).

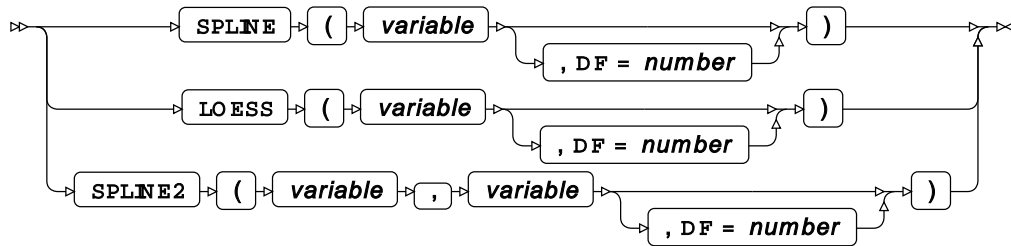
LABEL



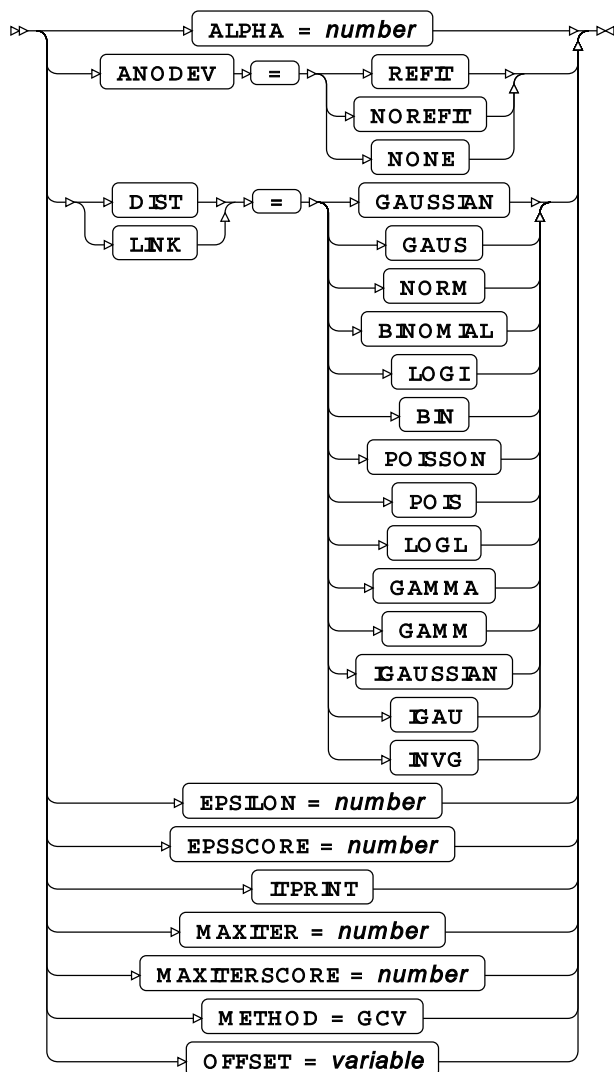
MODEL



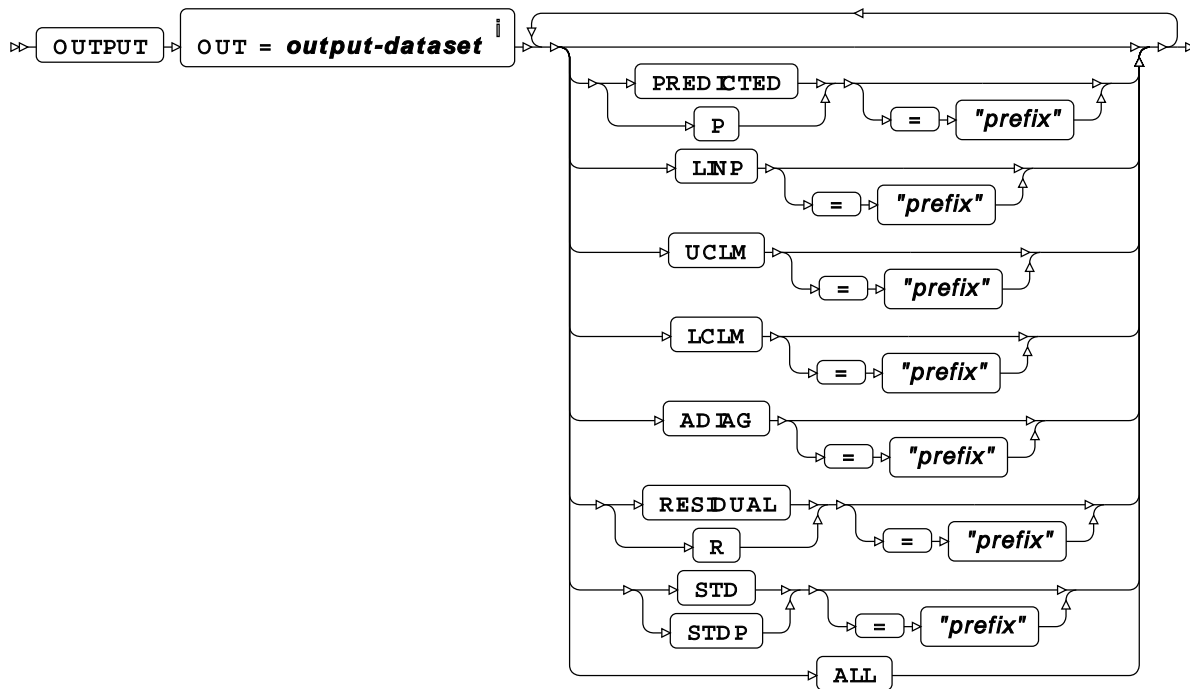
smoothing options



options



OUTPUT



ⁱ See *Output dataset* [\[link\]](#) (page 16).

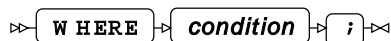
SCORE



ⁱ See *Input dataset* [\[link\]](#) (page 16).

ⁱⁱ See *Output dataset* [\[link\]](#) (page 16).

WHERE

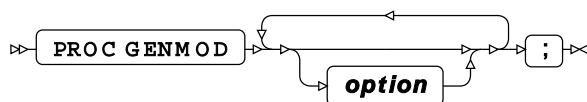


GENMOD procedure

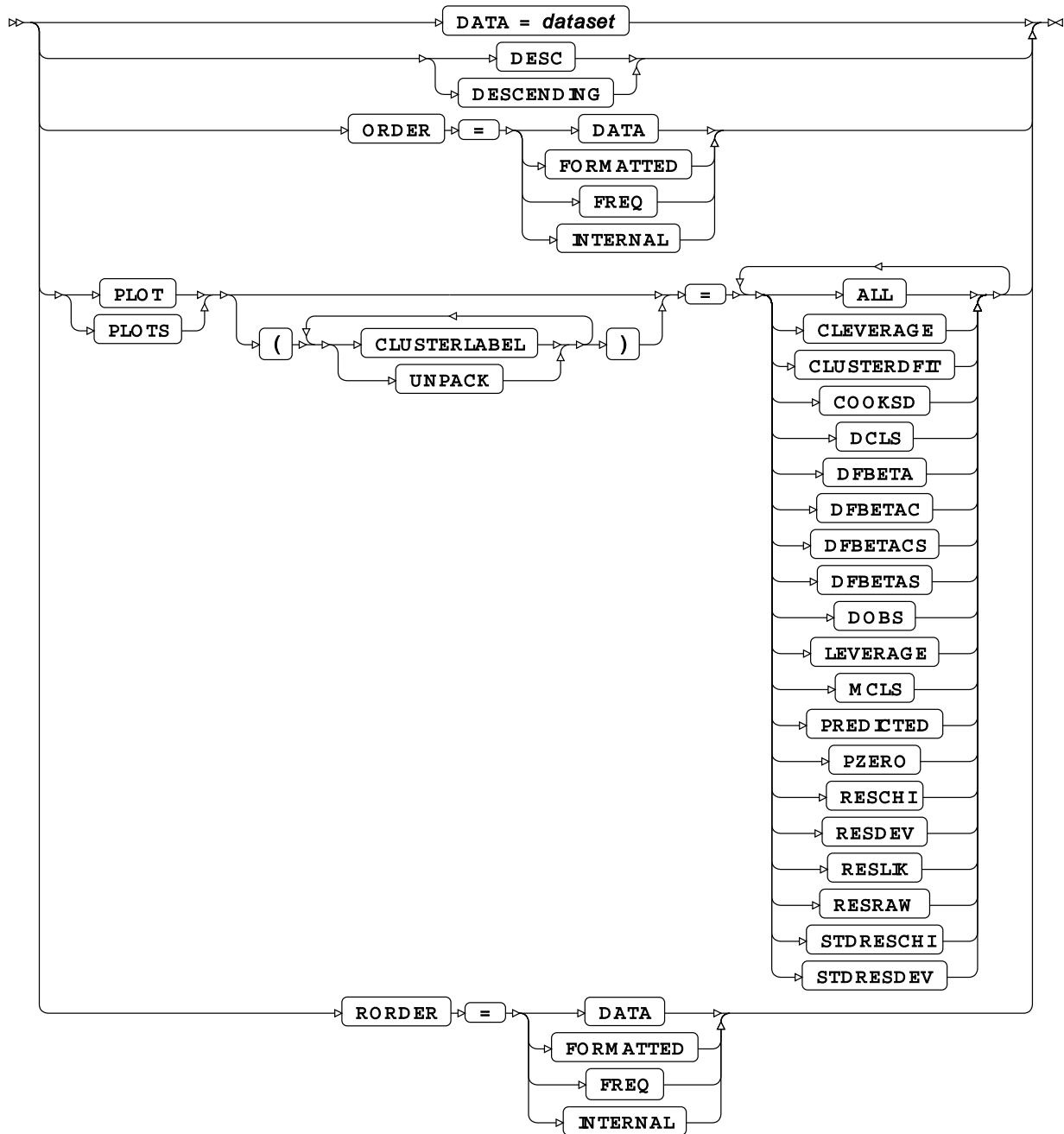
Supported statements

- *PROC GENMOD* [↗](#) (page 2861)
- *ATTRIB* [↗](#) (page 2863)
- *BY* [↗](#) (page 2863)
- *CLASS* [↗](#) (page 2863)
- *CODE* [↗](#) (page 2864)
- *CONTRAST* [↗](#) (page 2865)
- *ESTIMATE* [↗](#) (page 2866)
- *DEVIANCE* [↗](#) (page 2866)
- *FORMAT* [↗](#) (page 2866)
- *FWDLINK* [↗](#) (page 2866)
- *INFORMAT* [↗](#) (page 2867)
- *INVLINK* [↗](#) (page 2867)
- *FREQ* [↗](#) (page 2867)
- *LABEL* [↗](#) (page 2867)
- *MODEL* [↗](#) (page 2867)
- *OUTPUT* [↗](#) (page 2870)
- *REPEATED* [↗](#) (page 2872)
- *VARIANCE* [↗](#) (page 2872)
- *WEIGHT* [↗](#) (page 2872)
- *WHERE* [↗](#) (page 2872)
- *ZERO* [↗](#) (page 2873)

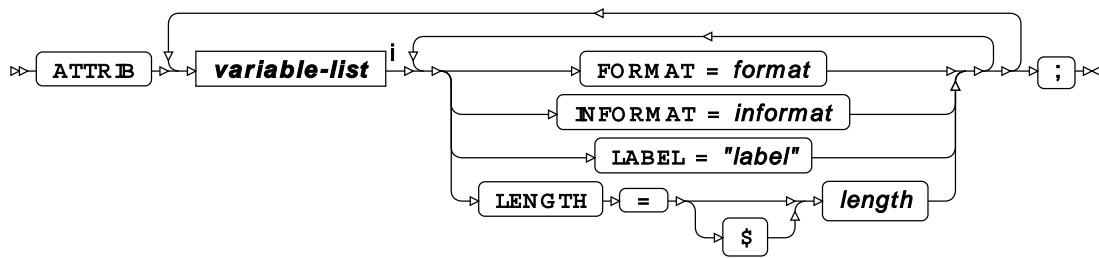
PROC GENMOD



option

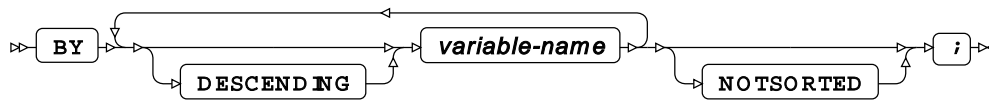


ATTRIB

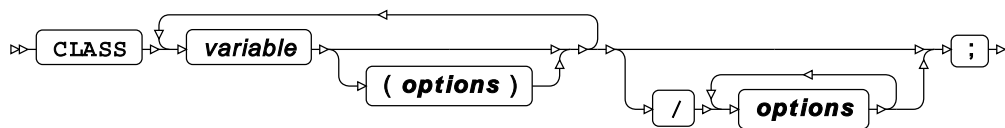


ⁱ See *Variable Lists* [↗](#) (page 32).

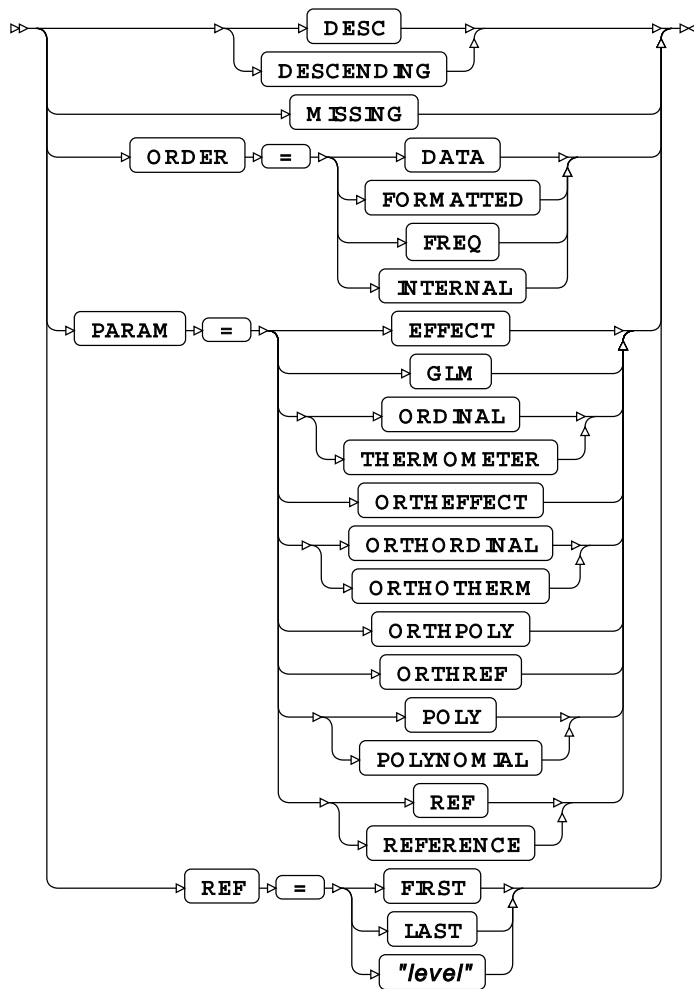
BY



CLASS



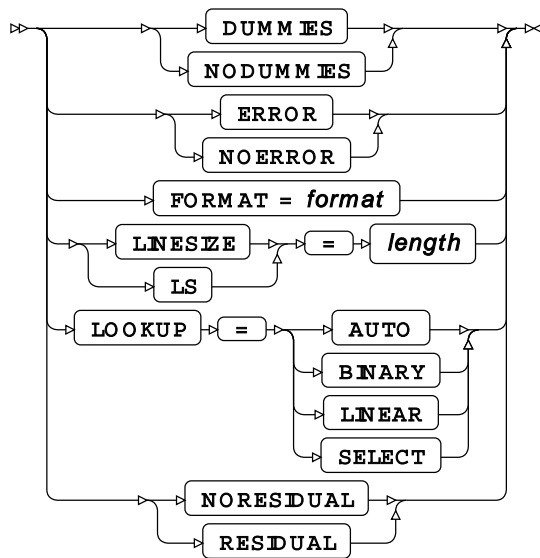
options



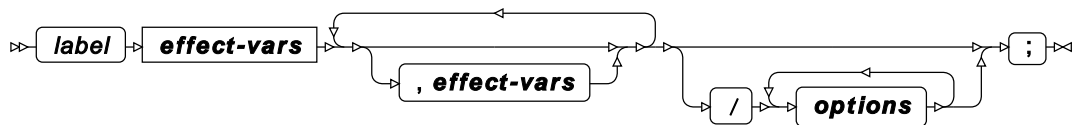
CODE



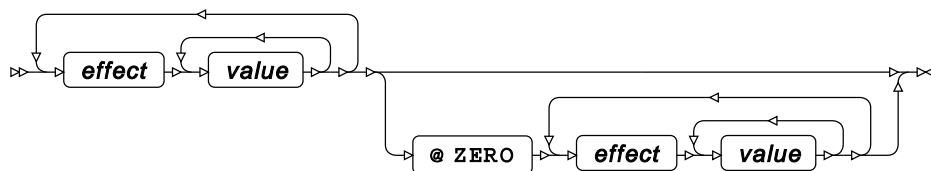
options



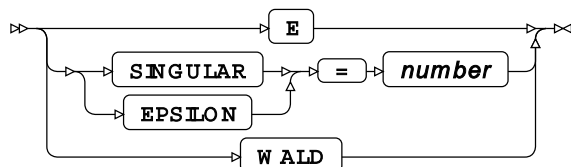
CONTRAST



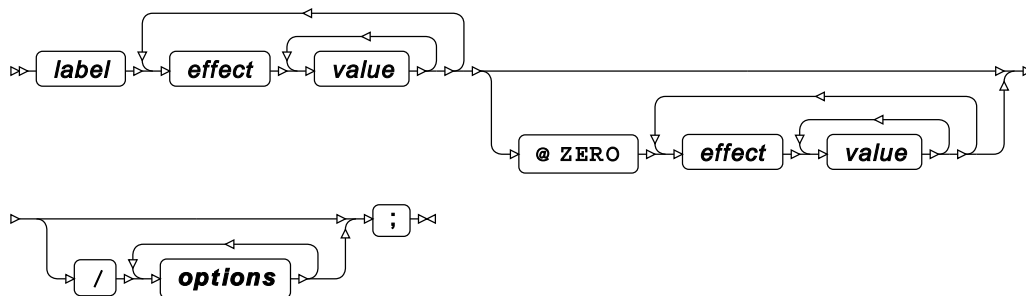
effect-vars



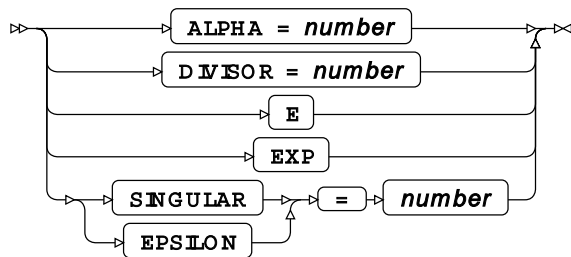
options



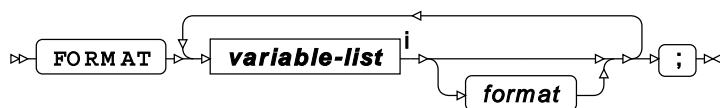
ESTIMATE



options

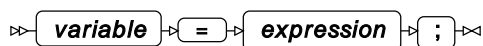


FORMAT

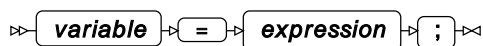


ⁱ See [Variable Lists](#) (page 32).

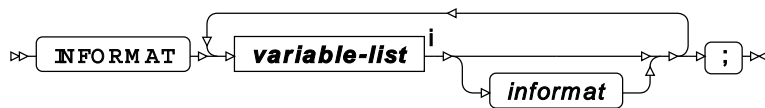
DEVIANCE



FWDLINK

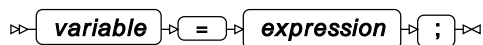


INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

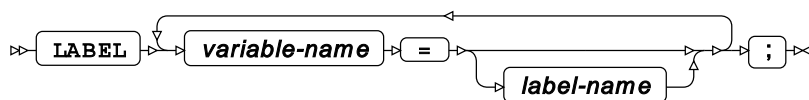
INVLINK



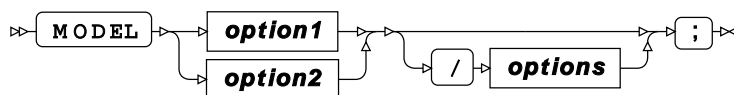
FREQ



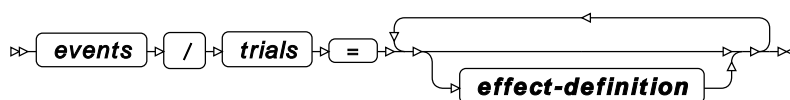
LABEL



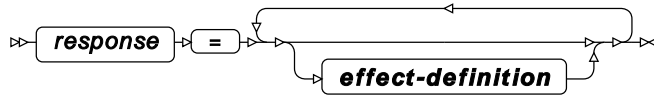
MODEL



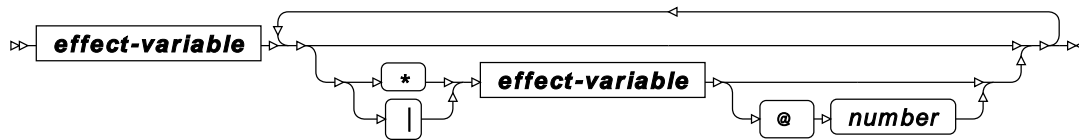
option1



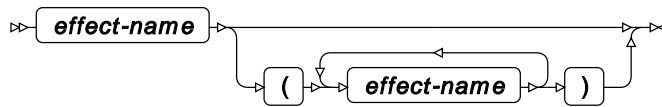
option2



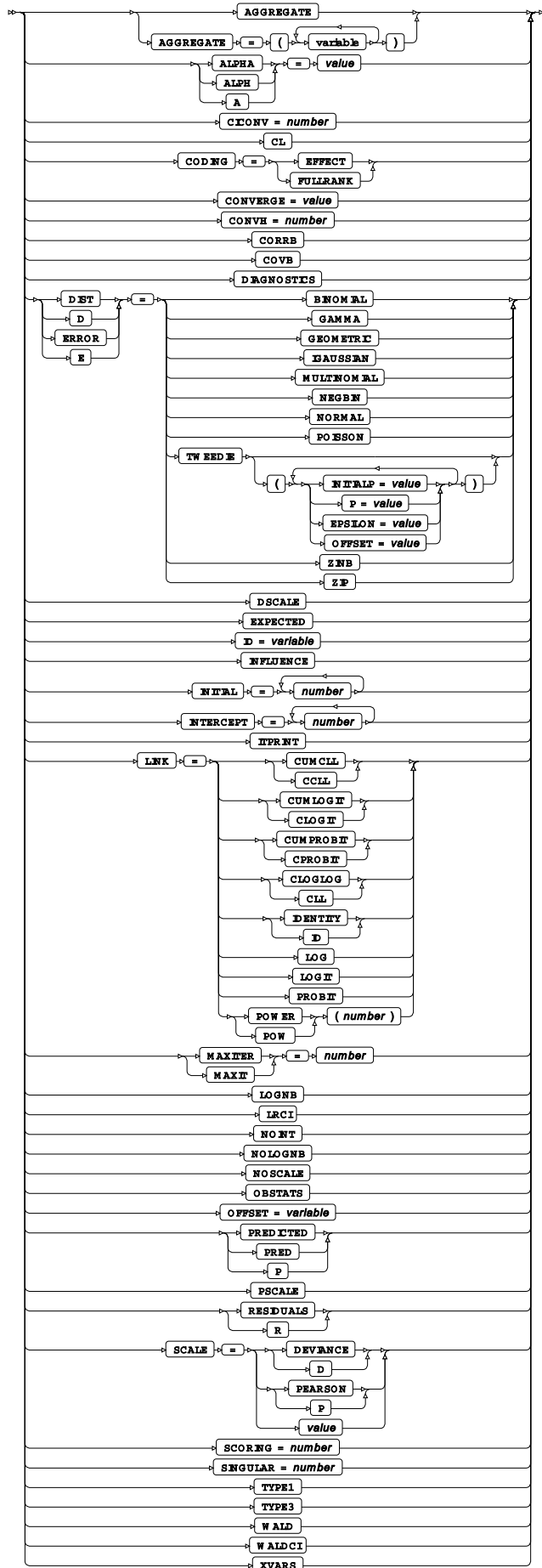
effect-definition



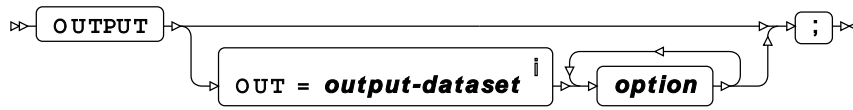
effect-variable



options

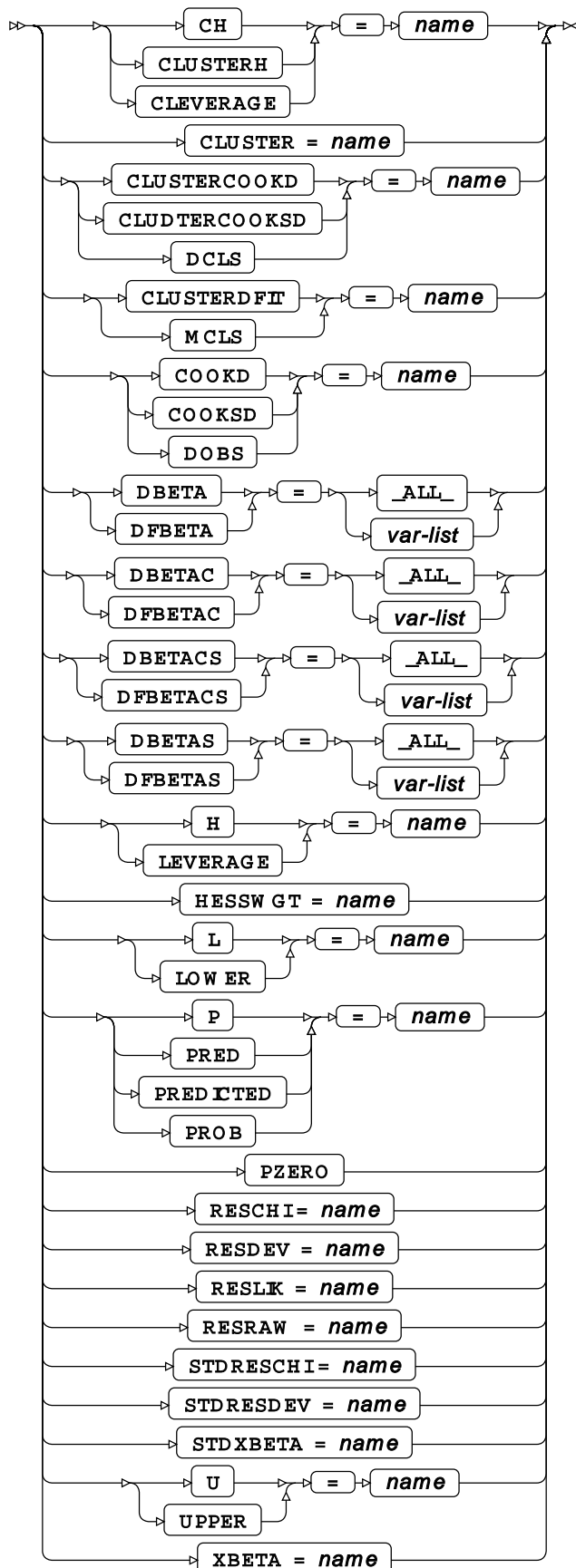


OUTPUT

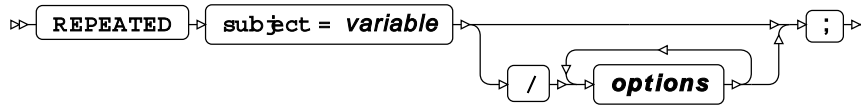


ⁱ See *Output dataset* [↗](#) (page 16).

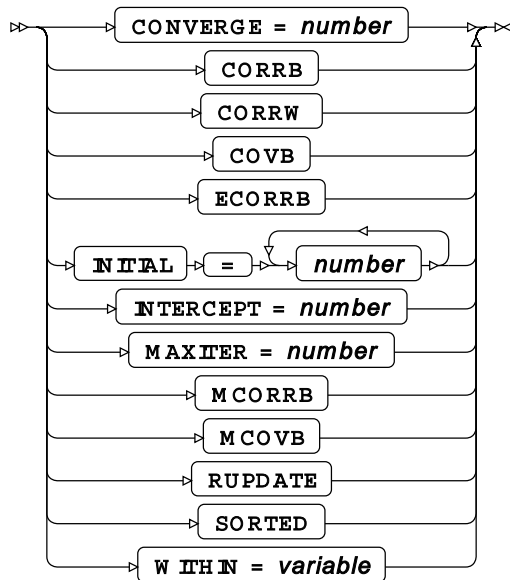
option



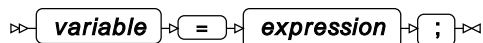
REPEATED



options



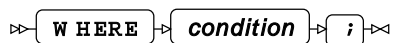
VARIANCE



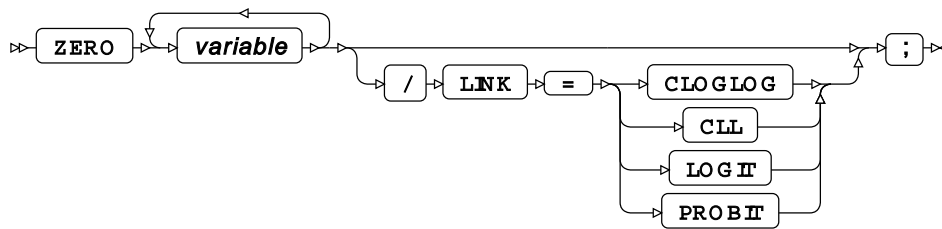
WEIGHT



WHERE



ZERO

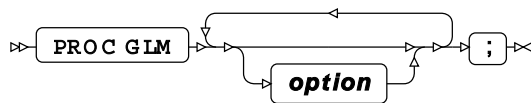


GLM procedure

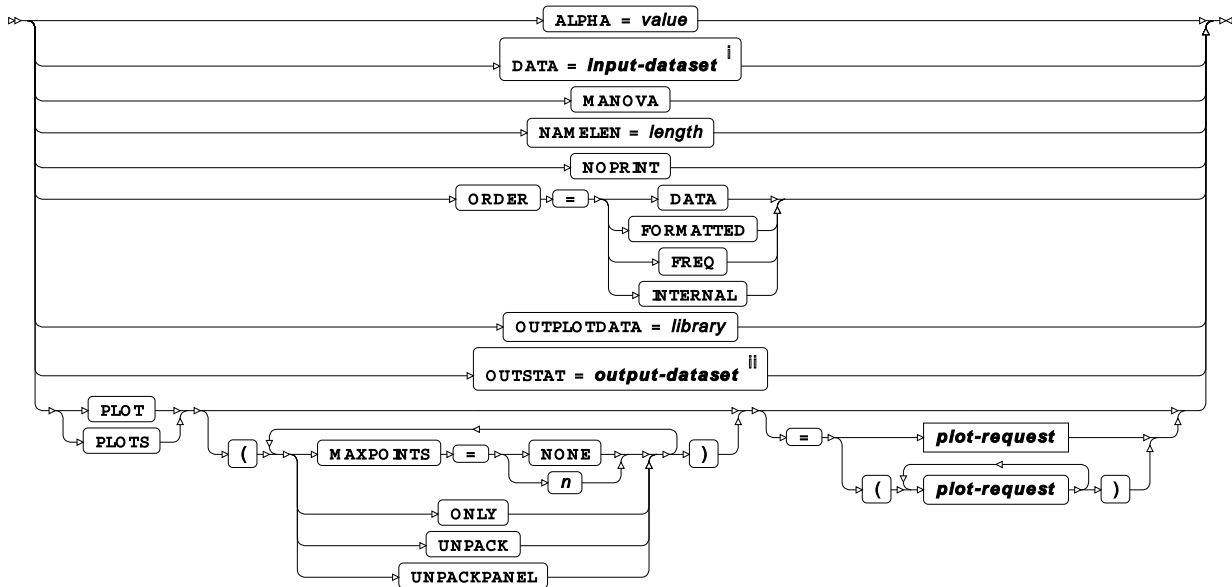
Supported statements

- *PROC GLM* [↗](#) (page 2874)
- *ATTRIB* [↗](#) (page 2876)
- *BY* [↗](#) (page 2876)
- *CLASS* [↗](#) (page 2876)
- *CODE* [↗](#) (page 2876)
- *CONTRAST* [↗](#) (page 2877)
- *ESTIMATE* [↗](#) (page 2878)
- *FORMAT* [↗](#) (page 2878)
- *FREQ* [↗](#) (page 2878)
- *INFORMAT* [↗](#) (page 2878)
- *LABEL* [↗](#) (page 2879)
- *LSMEANS* [↗](#) (page 2879)
- *MEANS* [↗](#) (page 2881)
- *MODEL* [↗](#) (page 2882)
- *OUTPUT* [↗](#) (page 2883)
- *RANDOM* [↗](#) (page 2884)
- *TEST* [↗](#) (page 2885)
- *WEIGHT* [↗](#) (page 2885)
- *WHERE* [↗](#) (page 2885)

PROC GLM



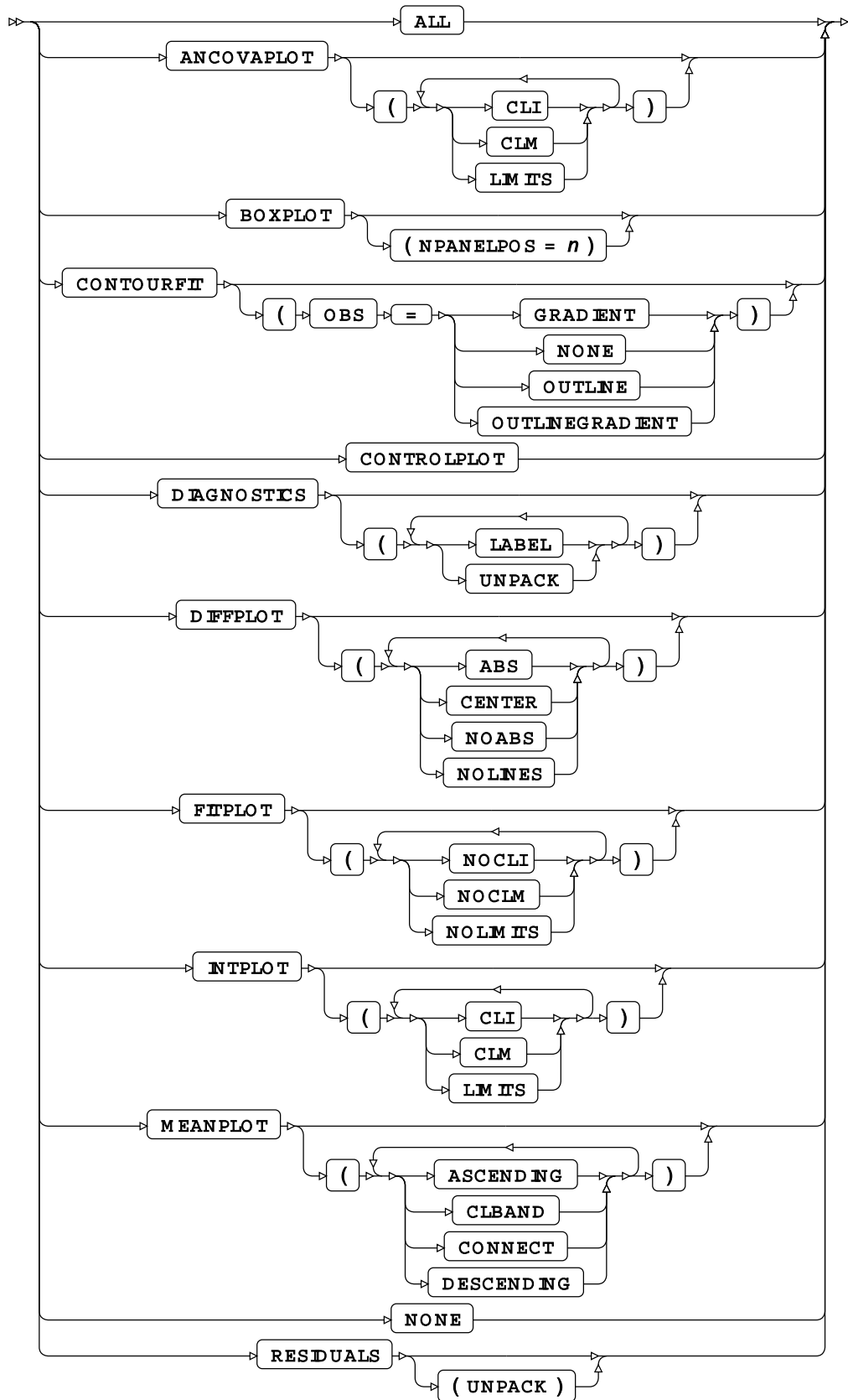
option



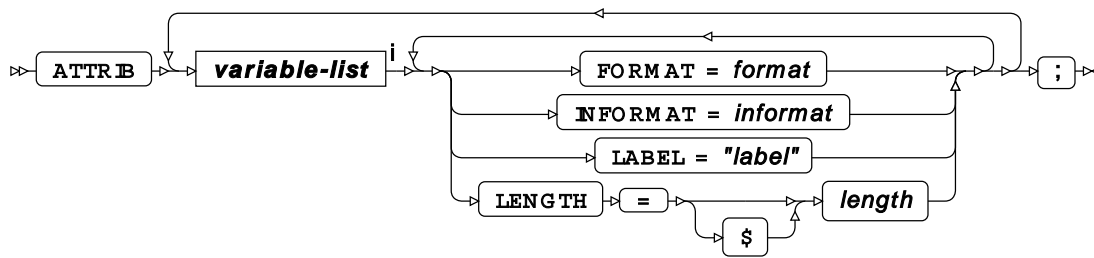
ⁱ See [Input dataset](#) (page 16).

ⁱⁱ See [Input dataset](#) (page 16).

plot-request

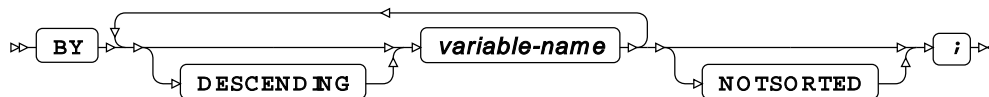


ATTRIB

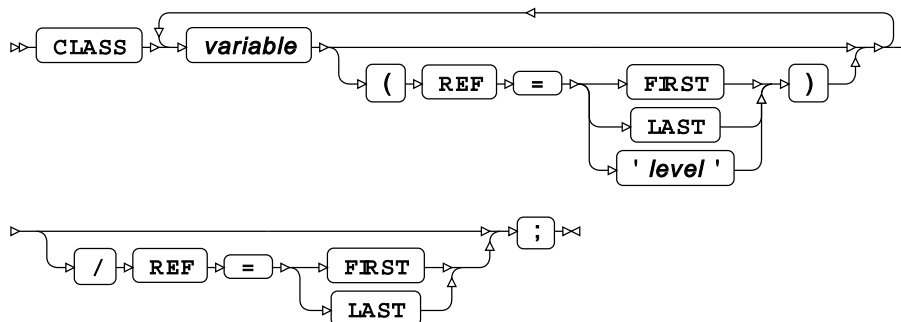


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



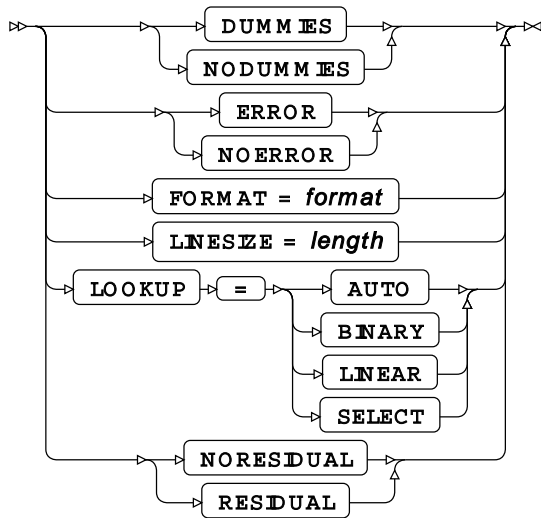
CLASS



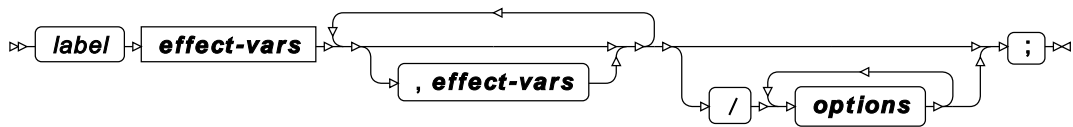
CODE



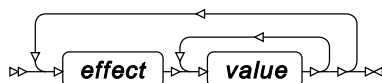
options



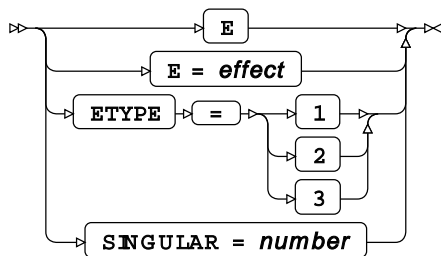
CONTRAST



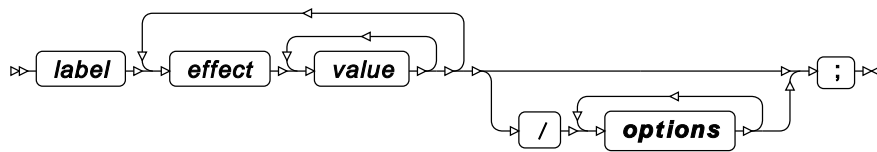
effect-vars



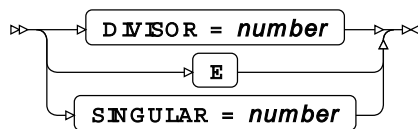
options



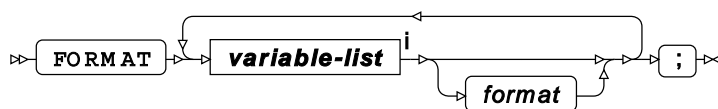
ESTIMATE



options



FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

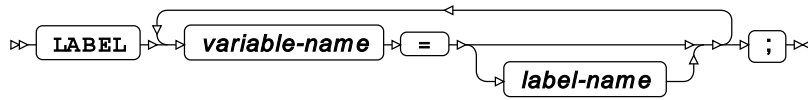


INFORMAT

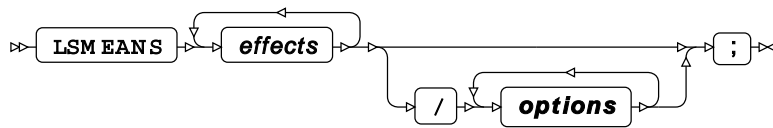


ⁱ See [Variable Lists](#) (page 32).

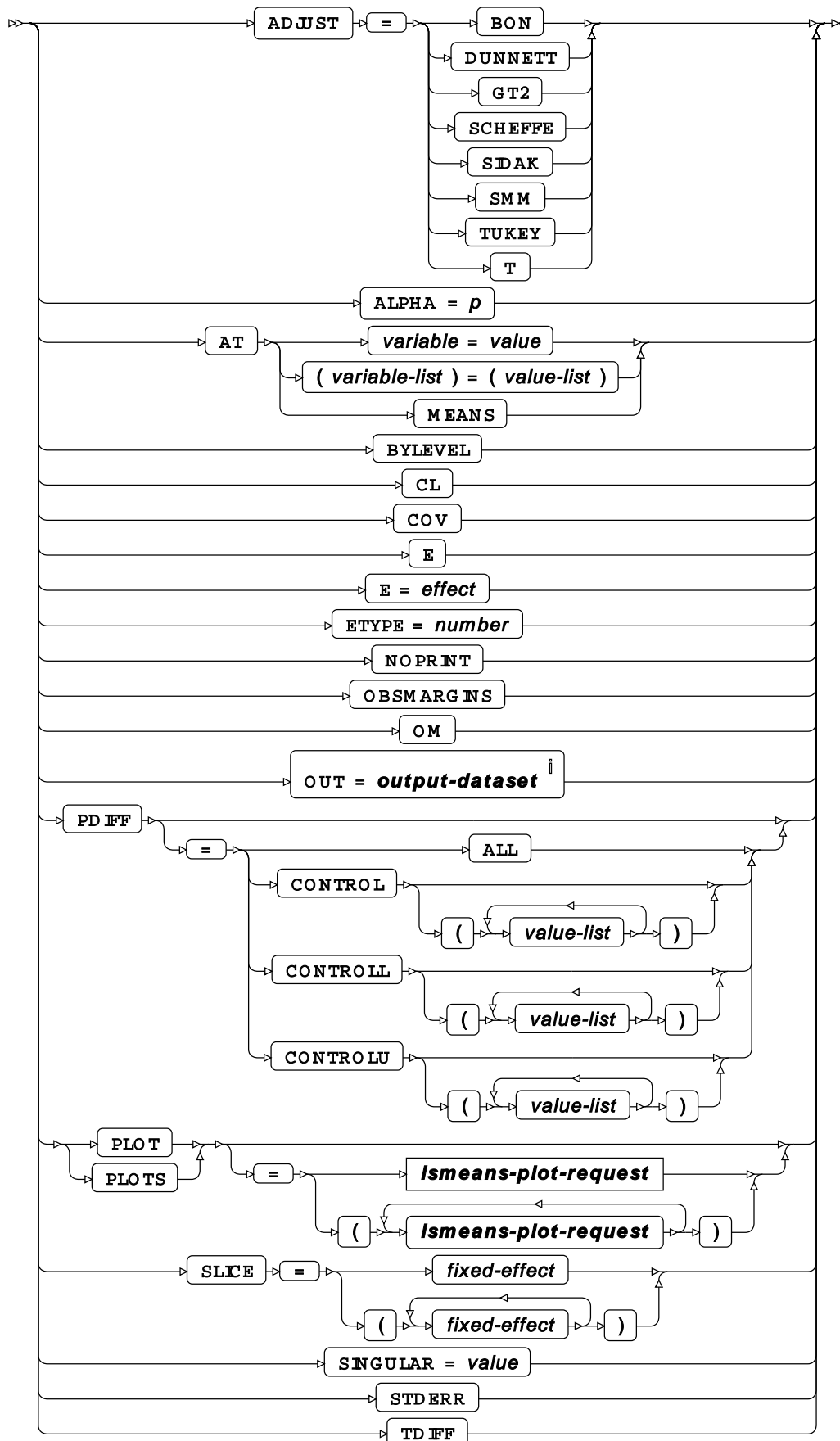
LABEL



LSMEANS

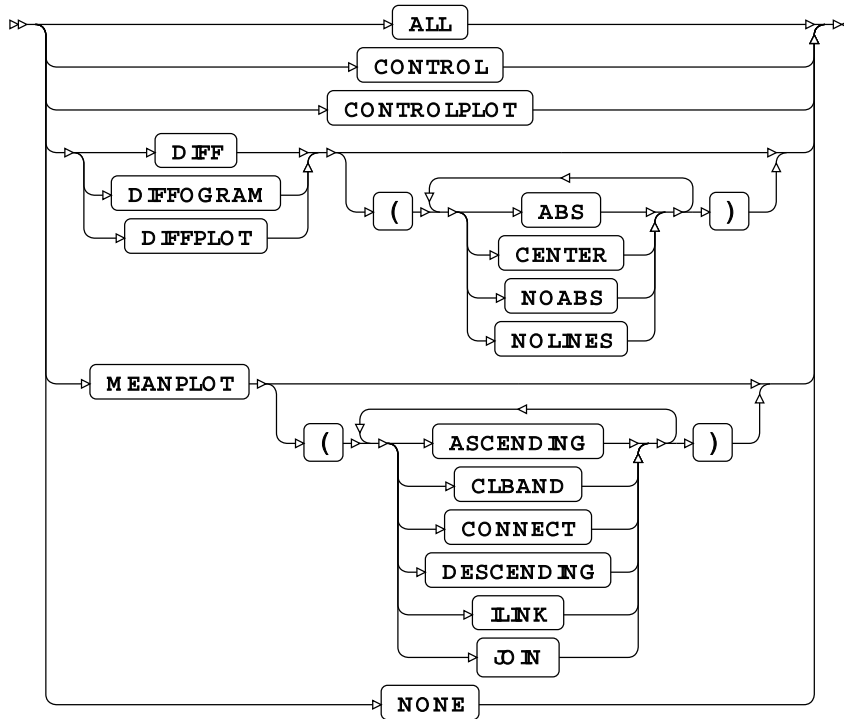


options

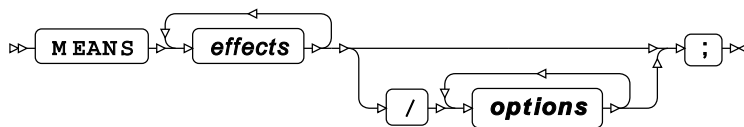


ⁱ See *Input dataset* [↗](#) (page 16).

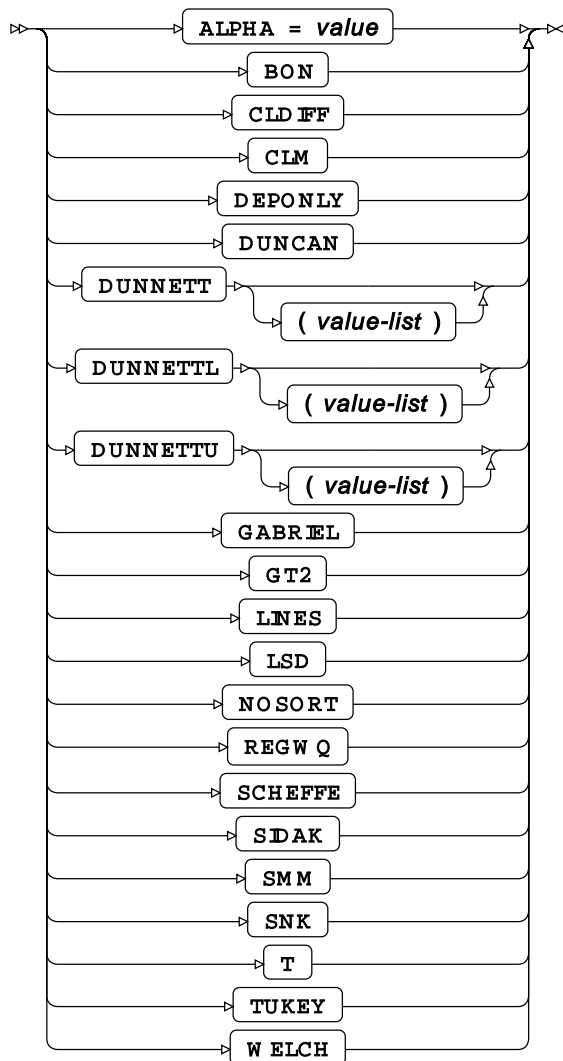
lsmeans-plot-request



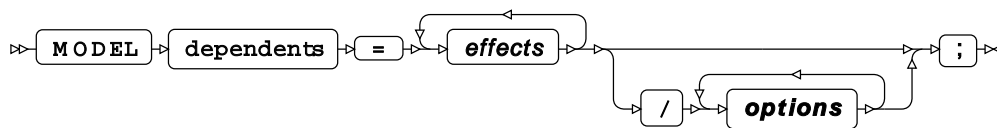
MEANS



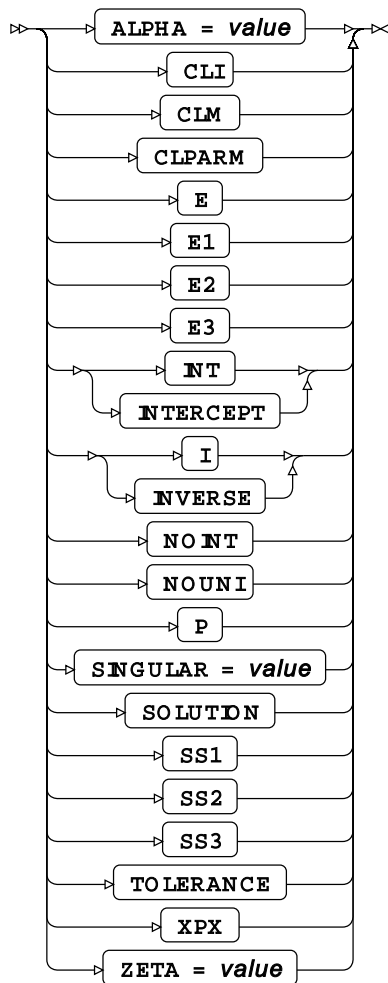
options



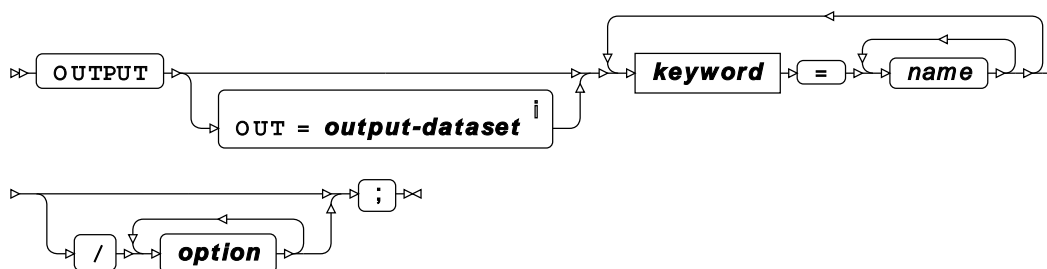
MODEL



options

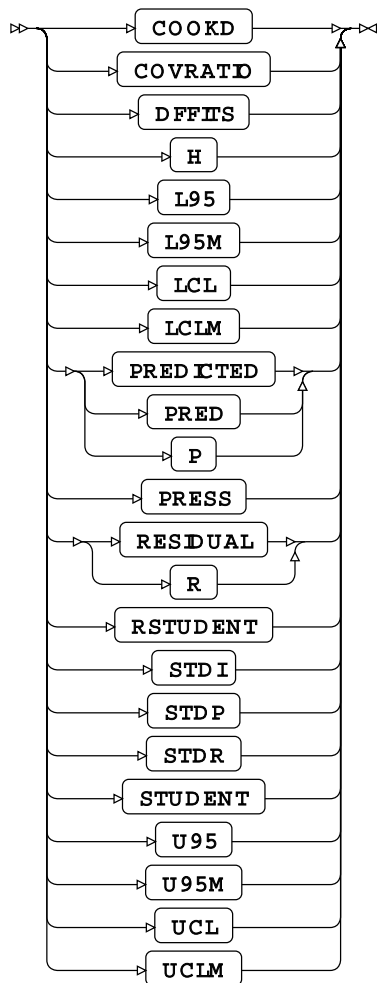


OUTPUT

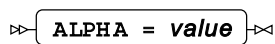


ⁱ See *Input dataset* [↗](#) (page 16).

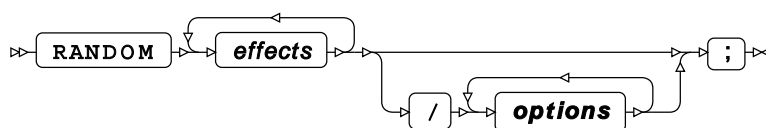
keyword



option



RANDOM



```

graph LR
    TEST[TEST] --> E[E]
    TEST --> H[H]
    E --> EQ1[=]
    EQ1 --> EF1[effect]
    H --> EQ2[=]
    EQ2 --> EF2[effect]
    EF1 --> DIV[/]
    EF2 --> DIV
    DIV --> ETYPE[ETYPE = n]
    DIV --> HTYPE[HTYPE = n]
    ETYPE --> SEMI[;]
    HTYPE --> SEMI
  
```

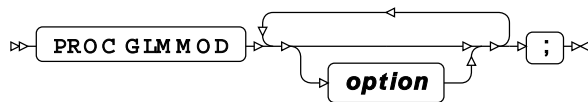
▷ **W E I G H T** ▷ *variable-name* ▷ **;** ▷

WHERE *condition* ;

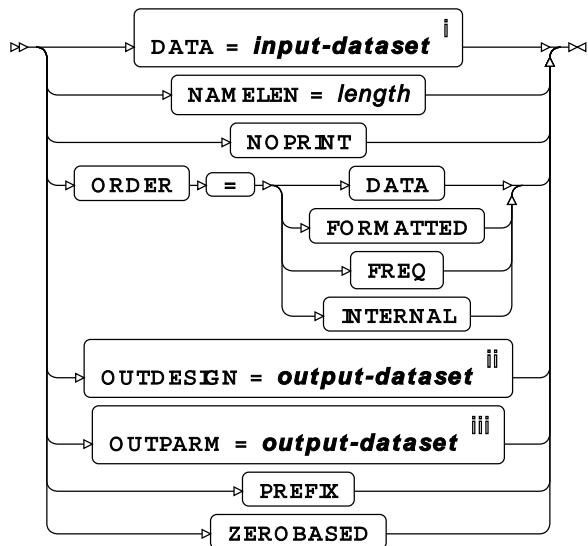
2885

- [MODEL](#) (page 2888)
- [WEIGHT](#) (page 2888)
- [WHERE](#) (page 2888)

PROC GLMMOD



option

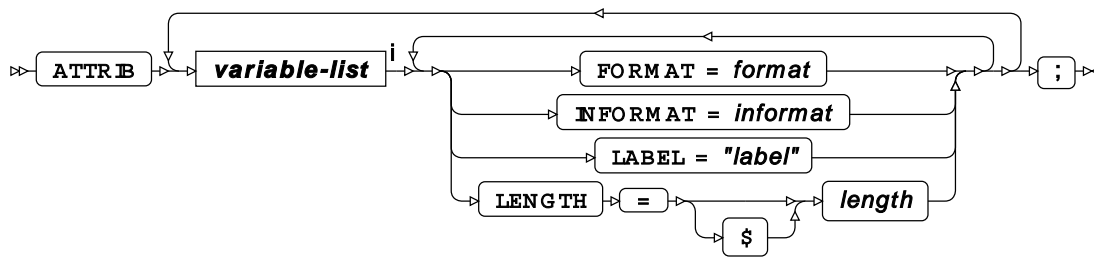


ⁱ See [Input dataset](#) (page 16).

ⁱⁱ See [Input dataset](#) (page 16).

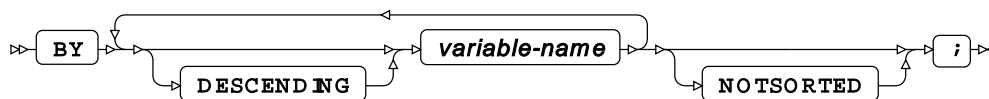
ⁱⁱⁱ See [Input dataset](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

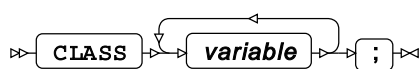


FORMAT

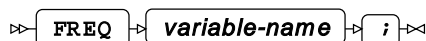


ⁱ See *Variable Lists* [↗](#) (page 32).

CLASS



FREQ

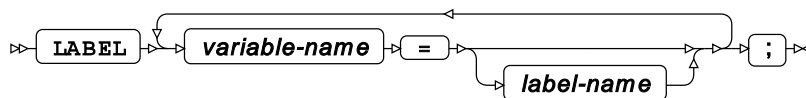


INFORMAT

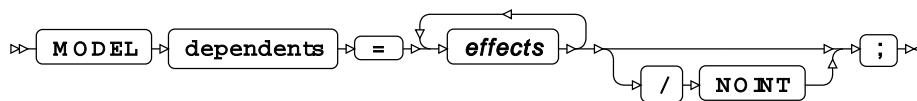


ⁱ See *Variable Lists* [↗](#) (page 32).

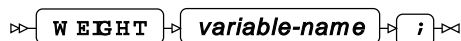
LABEL



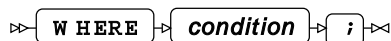
MODEL



WEIGHT



WHERE



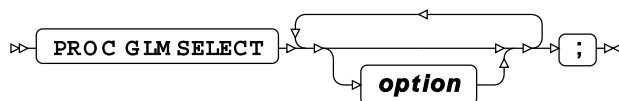
GLMSELECT procedure

Supported statements

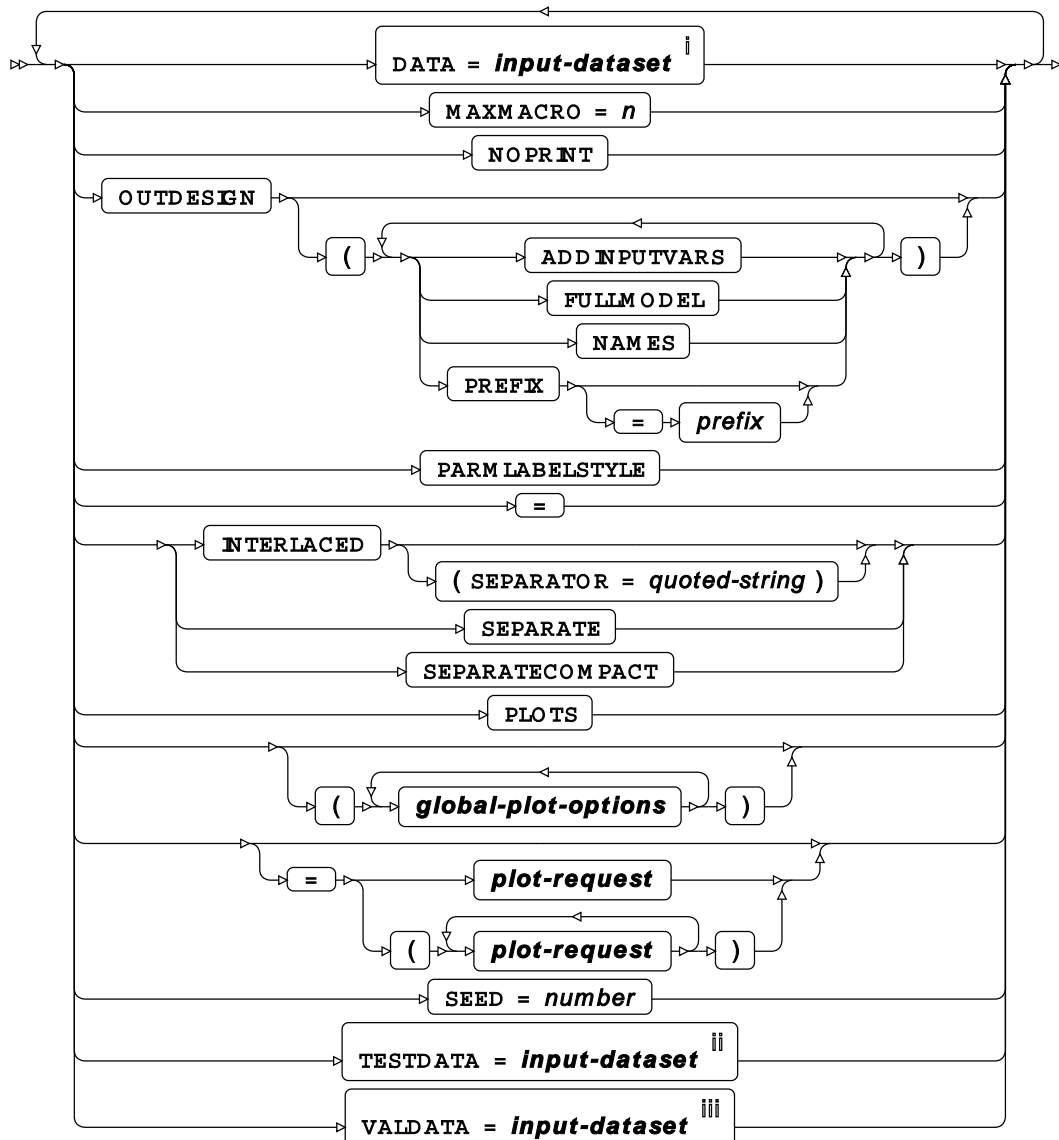
- *PROC GLMSELECT* [↗](#) (page 2889)

- *ATTRIB* [↗](#) (page 2892)
- *BY* [↗](#) (page 2892)
- *CLASS* [↗](#) (page 2892)
- *CODE* [↗](#) (page 2894)
- *FORMAT* [↗](#) (page 2895)
- *FREQ* [↗](#) (page 2895)
- *INFORMAT* [↗](#) (page 2895)
- *LABEL* [↗](#) (page 2896)
- *MODEL* [↗](#) (page 2896)
- *OUTPUT* [↗](#) (page 2899)
- *SCORE* [↗](#) (page 2899)
- *WEIGHT* [↗](#) (page 2900)

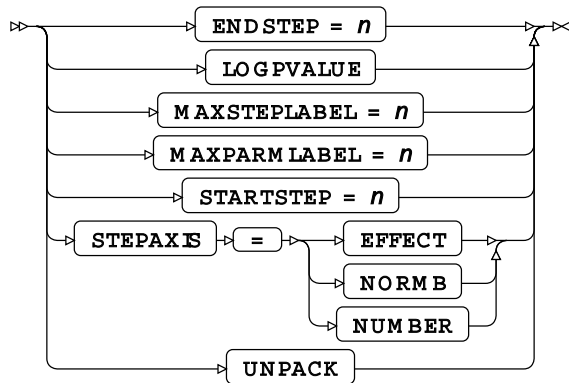
PROC GLMSELECT



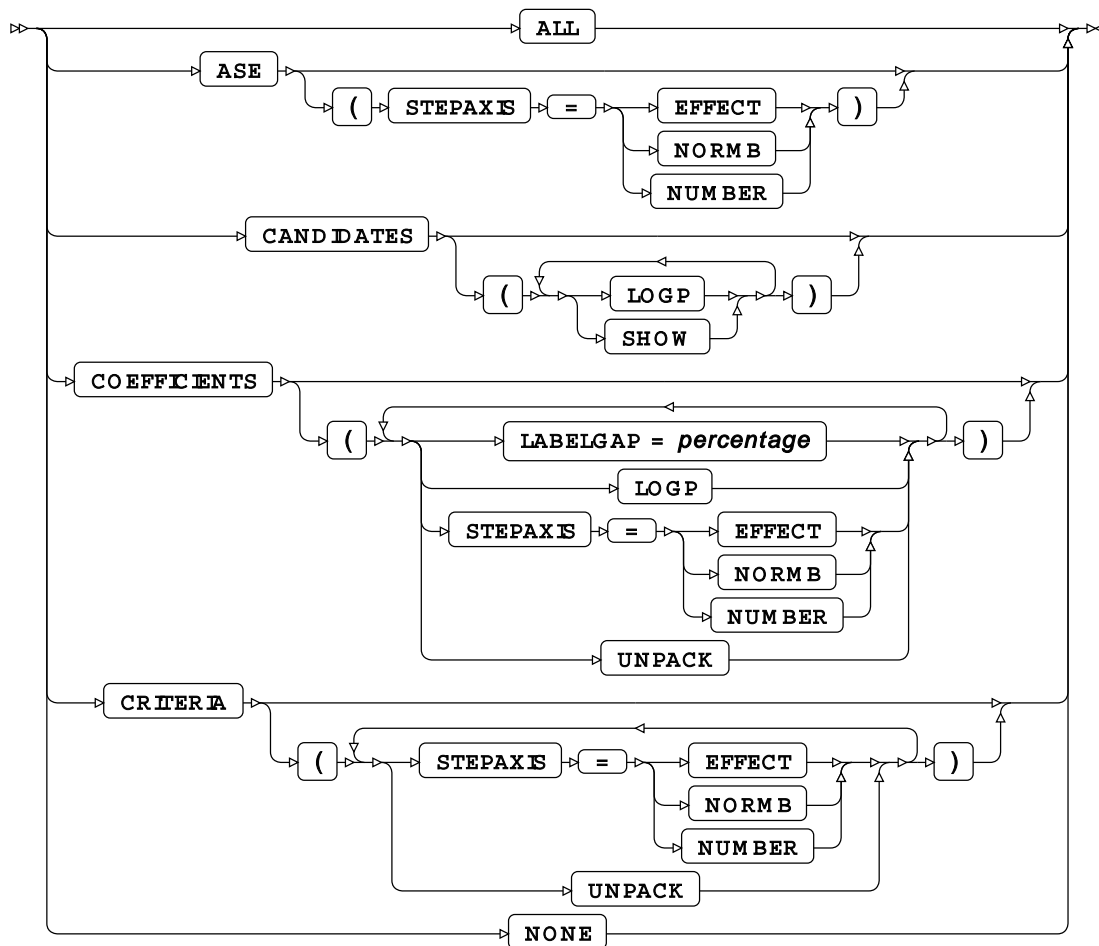
option

ⁱ See *Input dataset* [↗](#) (page 16).ⁱⁱ See *Input dataset* [↗](#) (page 16).ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

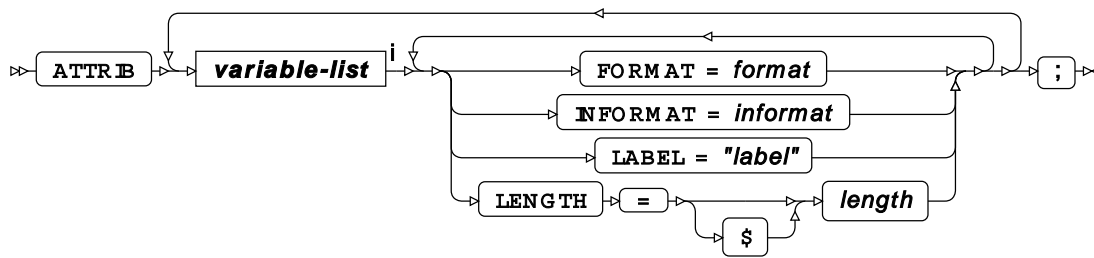
global-plot-options



plot-request

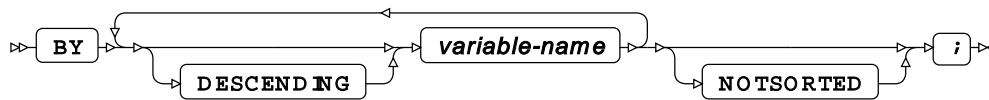


ATTRIB

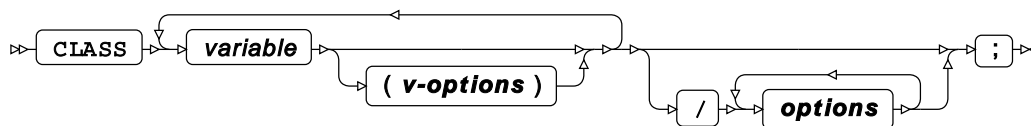


ⁱ See *Variable Lists* [↗](#) (page 32).

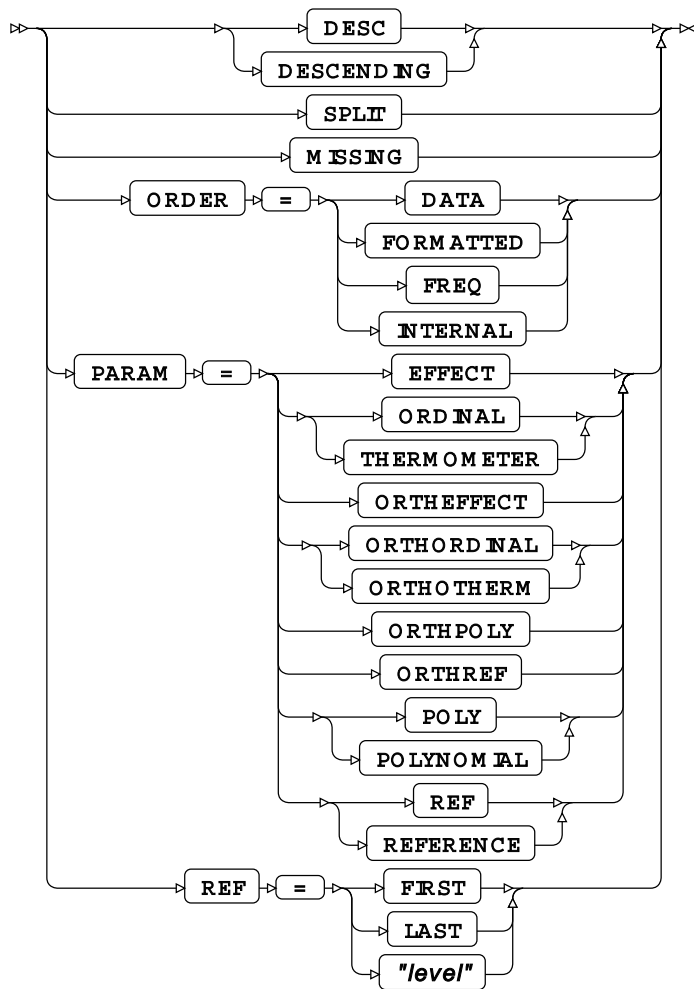
BY



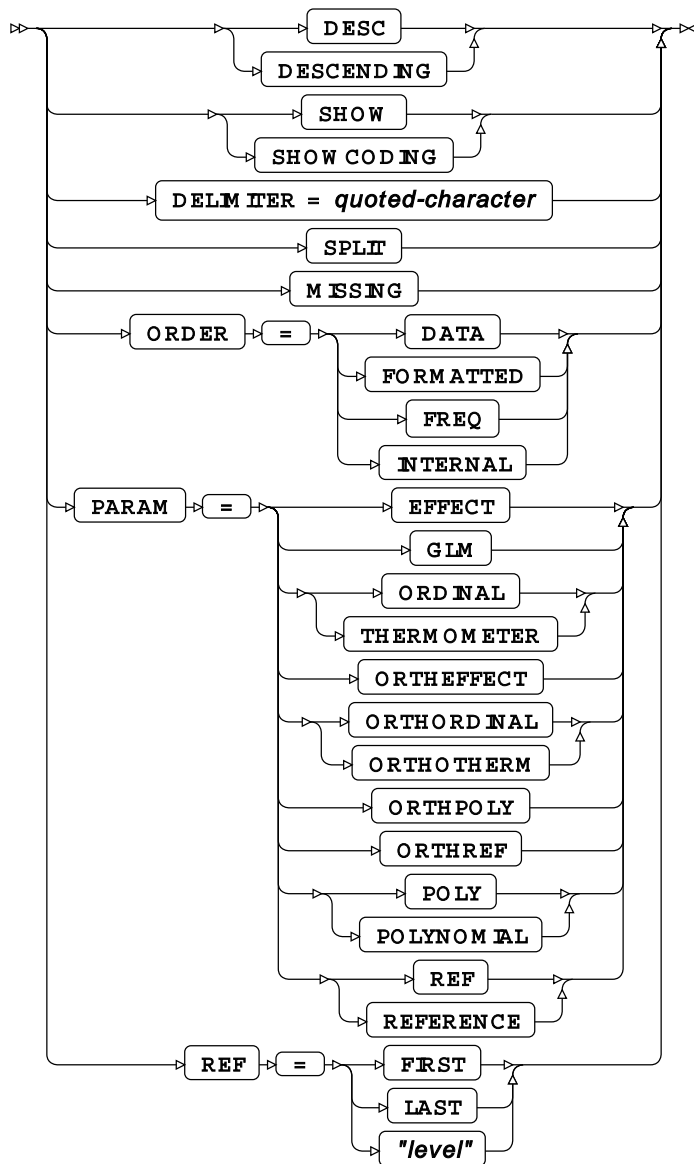
CLASS



v-options



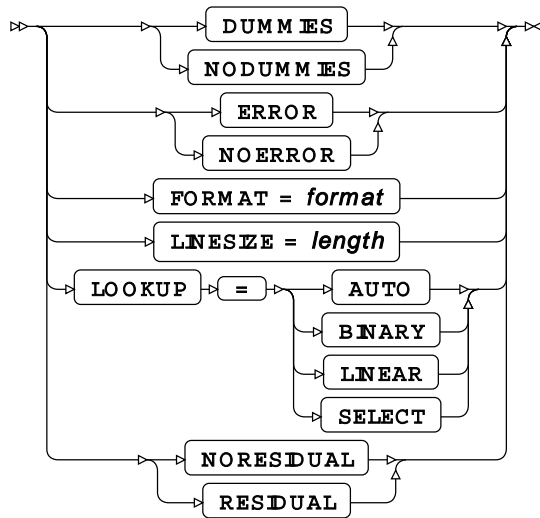
options



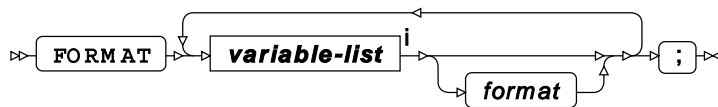
CODE



options



FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

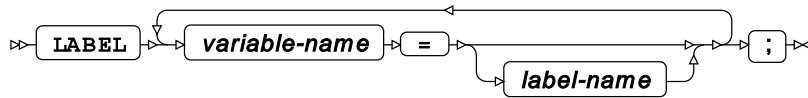


INFORMAT

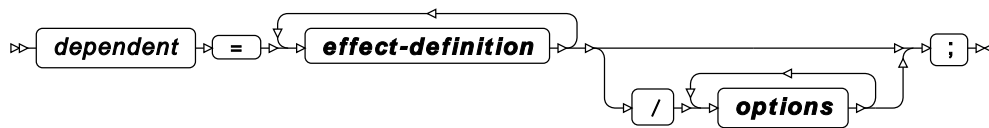


ⁱ See [Variable Lists](#) (page 32).

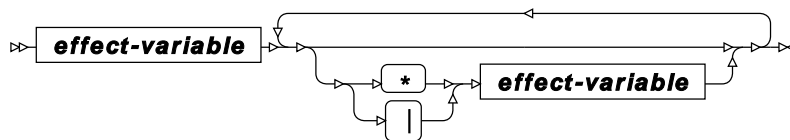
LABEL



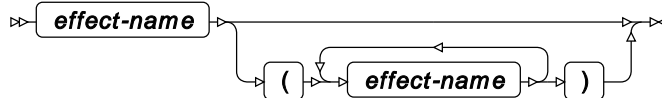
MODEL



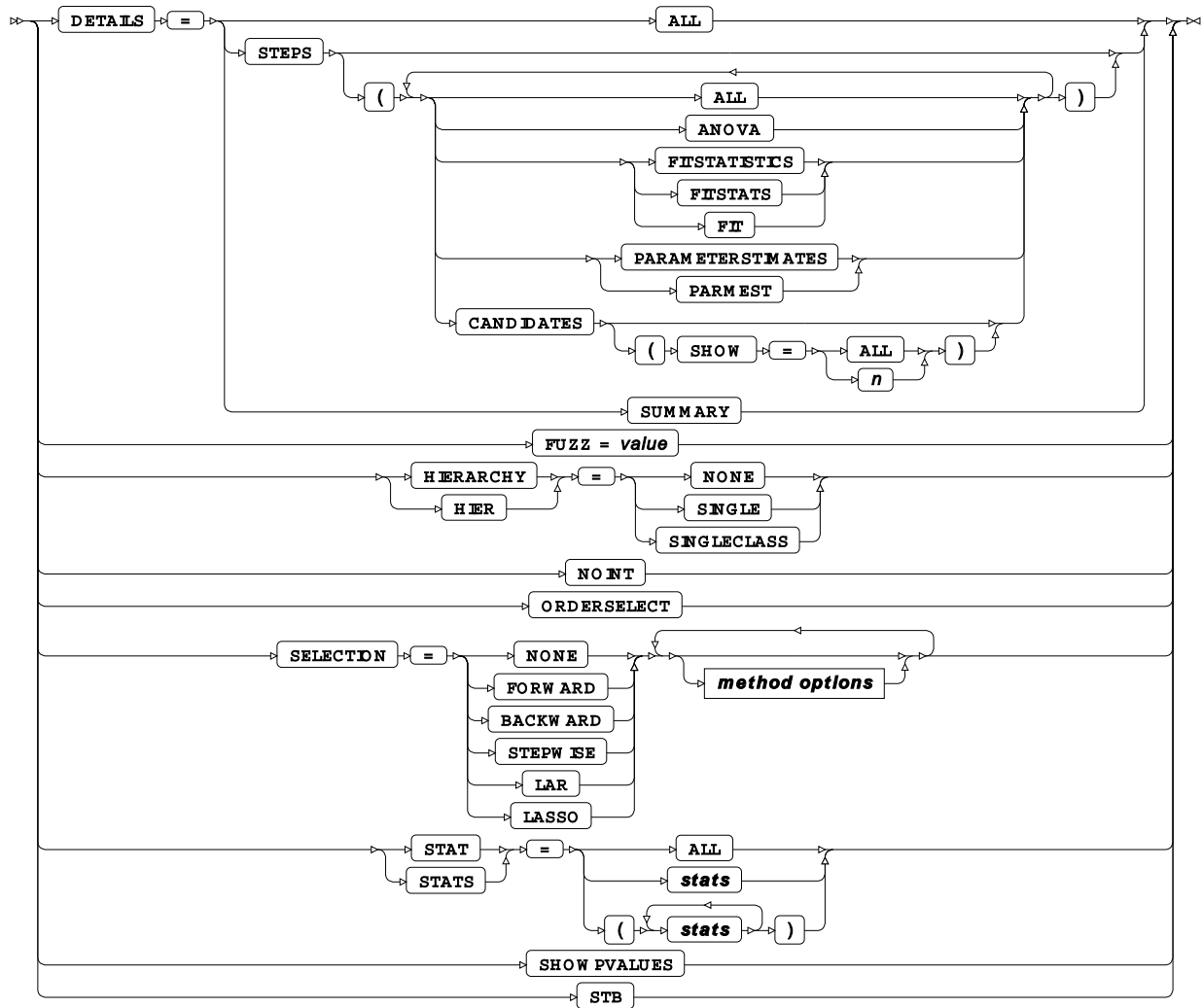
effect-definition



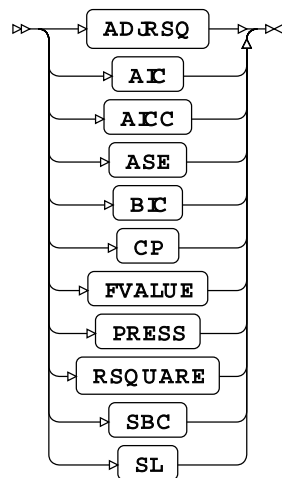
effect-variable



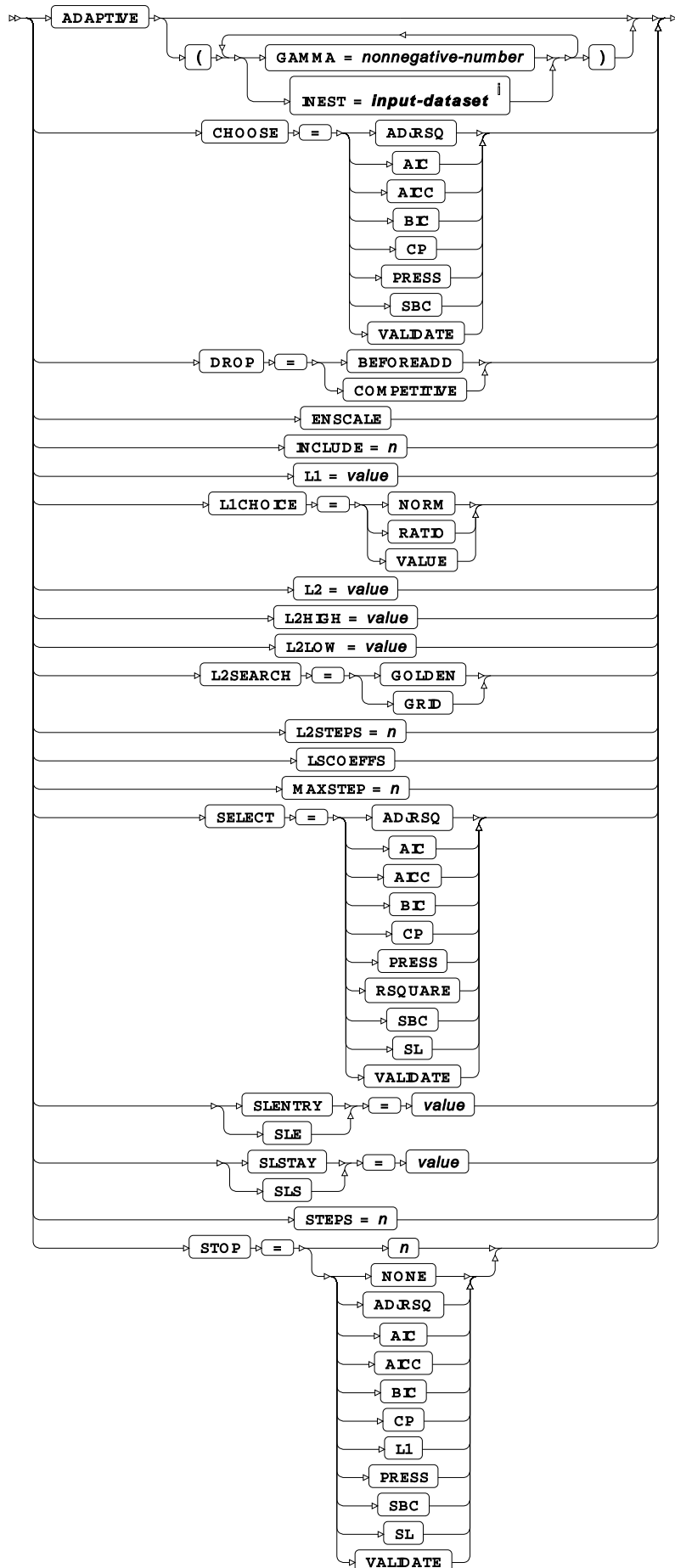
options



stats

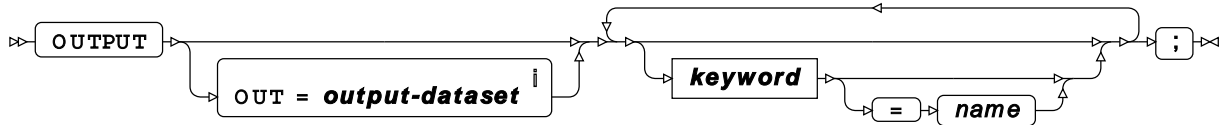


method options



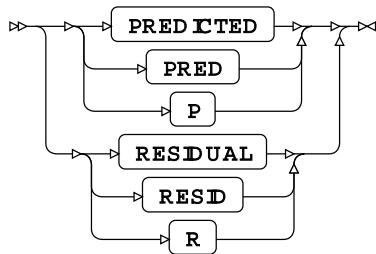
ⁱ See *Input dataset* [↗](#) (page 16).

OUTPUT

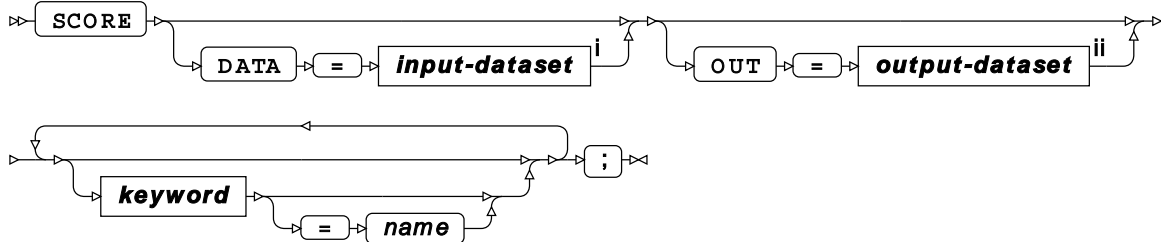


ⁱ See *Output dataset* [↗](#) (page 16).

keyword



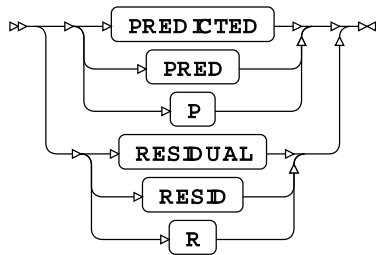
SCORE



ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

keyword



WEIGHT

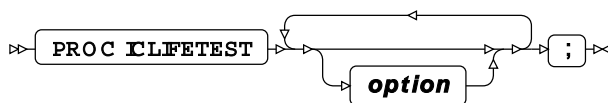


ICLIFETEST procedure

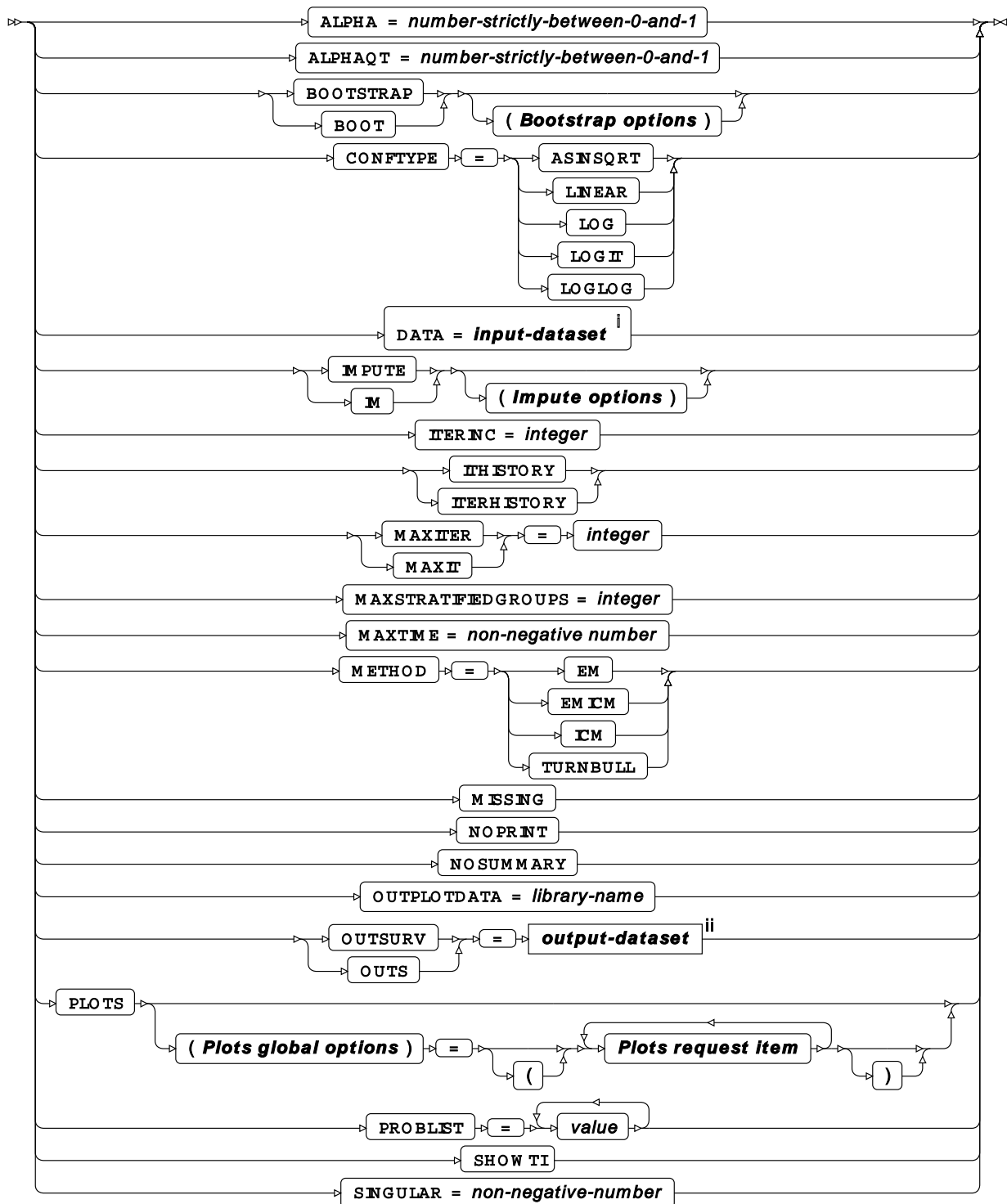
Supported statements

- *PROC ICLIFETEST* [↗](#) (page 2900)
- *ATTRIB* [↗](#) (page 2904)
- *BY* [↗](#) (page 2904)
- *FORMAT* [↗](#) (page 2904)
- *FREQ* [↗](#) (page 2905)
- *INFORMAT* [↗](#) (page 2905)
- *LABEL* [↗](#) (page 2905)
- *STRATA* [↗](#) (page 2905)
- *TEST* [↗](#) (page 2906)
- *TIME* [↗](#) (page 2906)
- *WHERE* [↗](#) (page 2906)

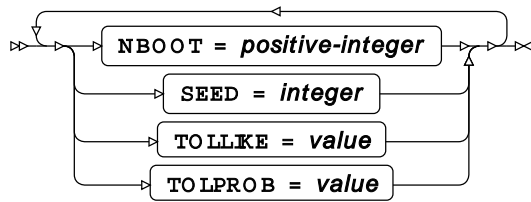
PROC ICLIFETEST



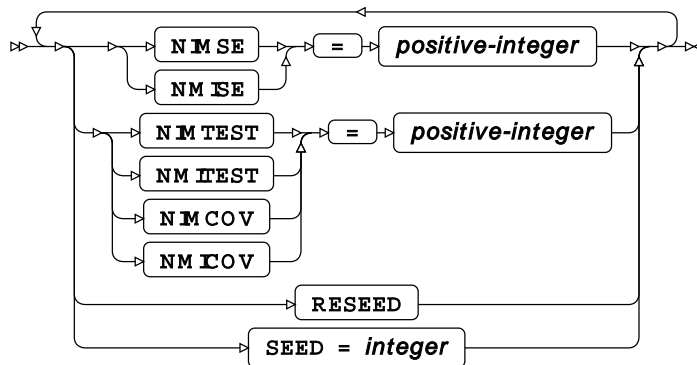
option

ⁱ See [Input dataset](#) (page 16).ⁱⁱ See [Input dataset](#) (page 16).

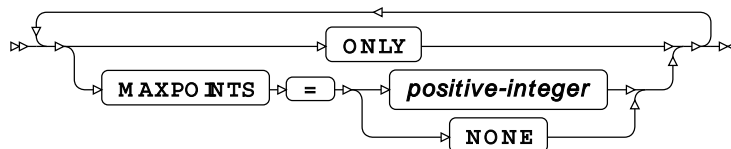
Bootstrap options



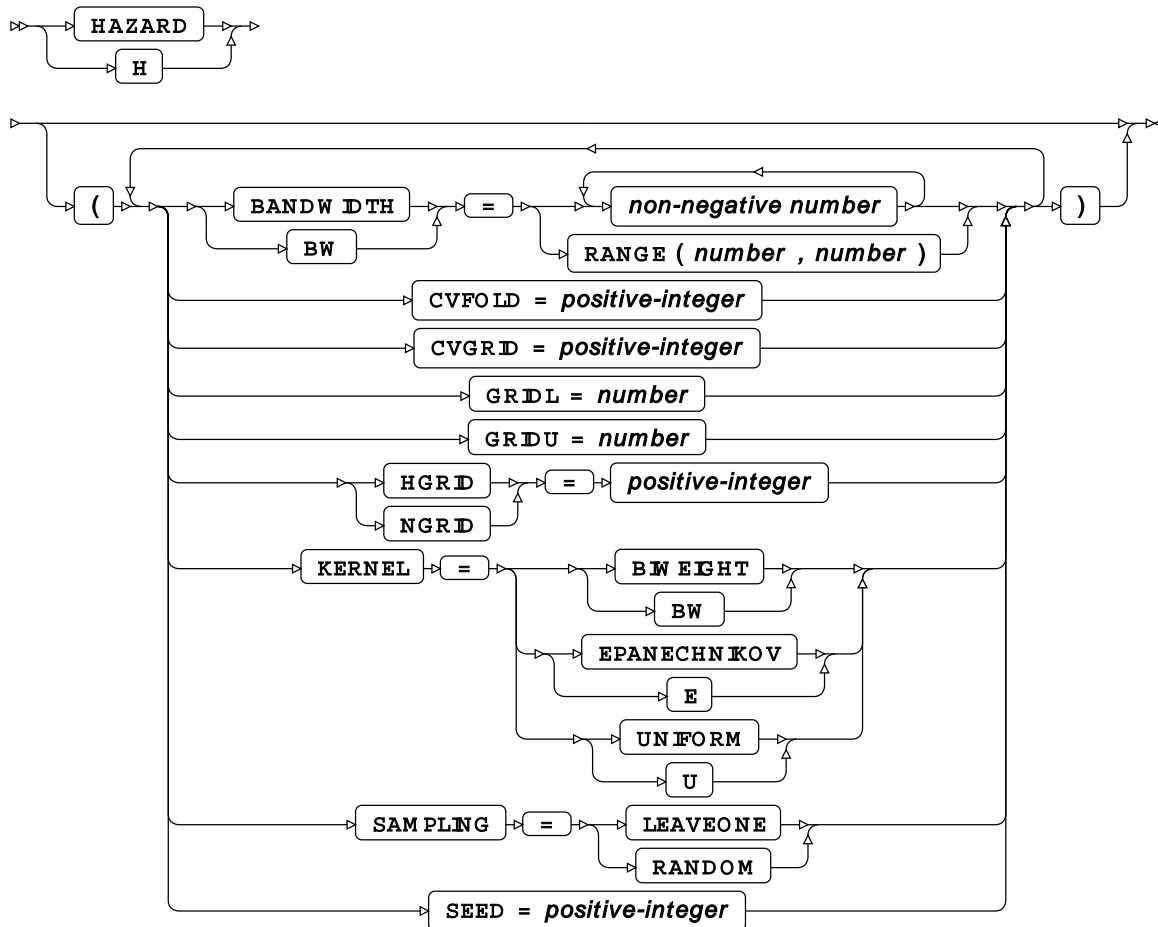
Impute options



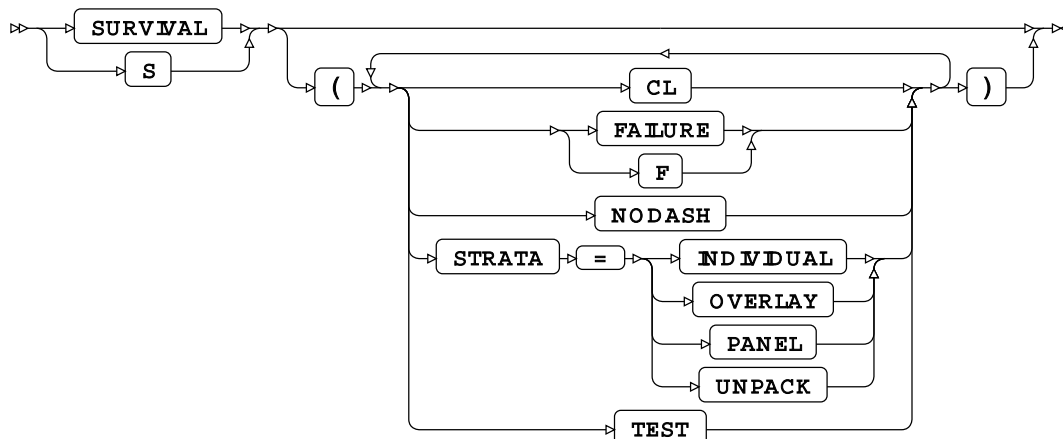
Plots global options



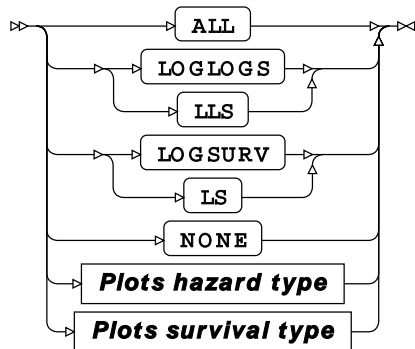
Plots hazard type



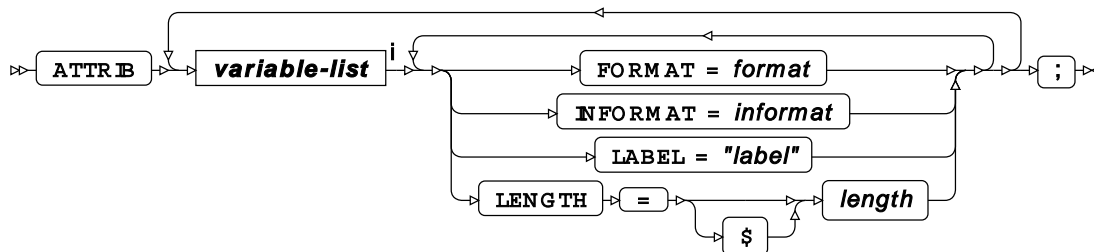
Plots survival type



Plots request item

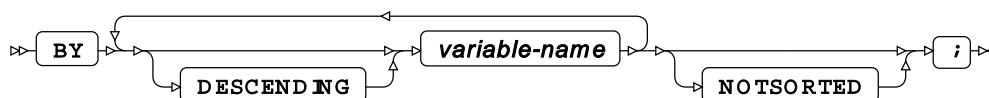


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

BY

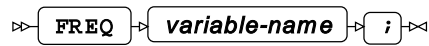


FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

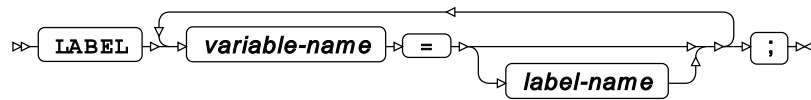


INFORMAT

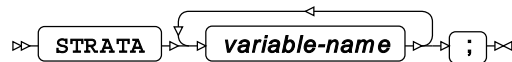


ⁱ See [Variable Lists](#) (page 32).

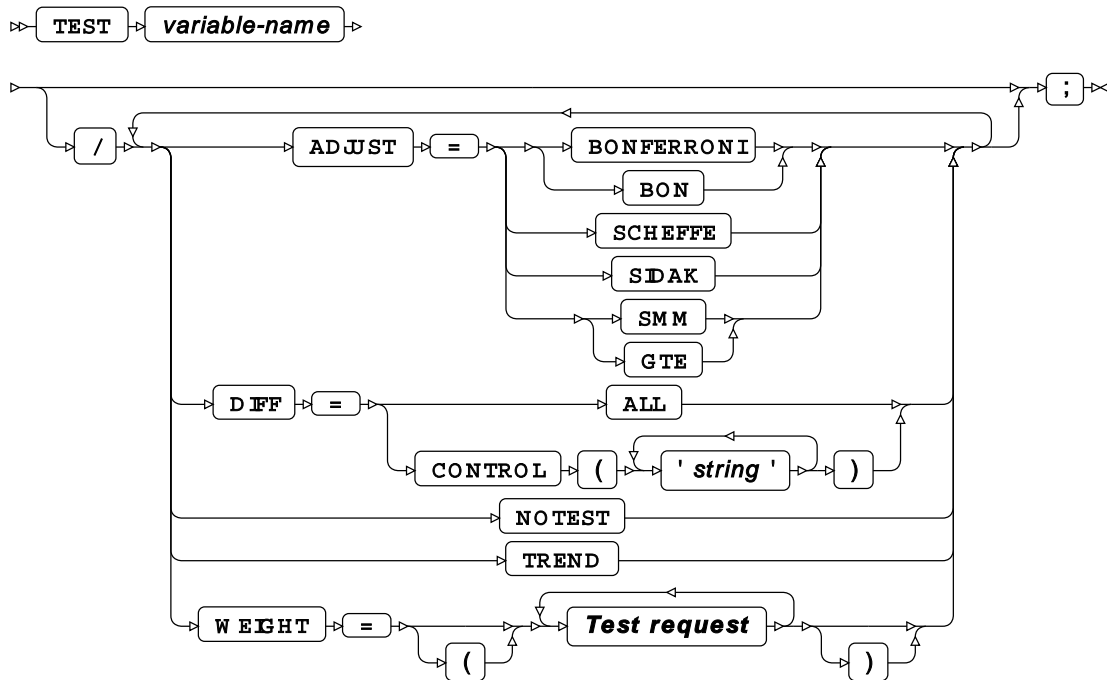
LABEL



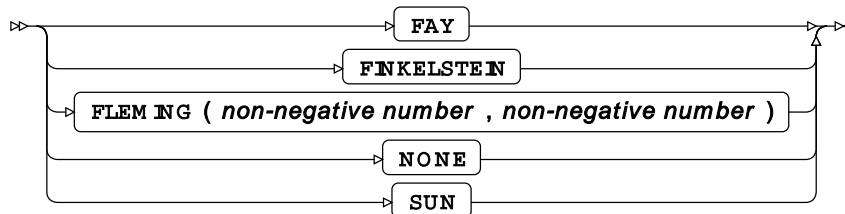
STRATA



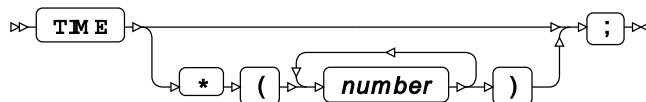
TEST



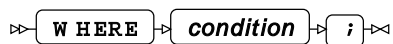
Test request



TIME



WHERE

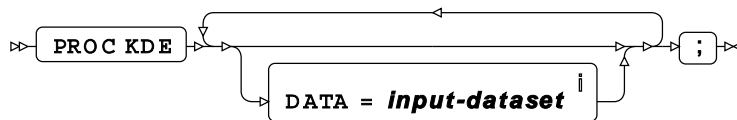


KDE procedure

Supported statements

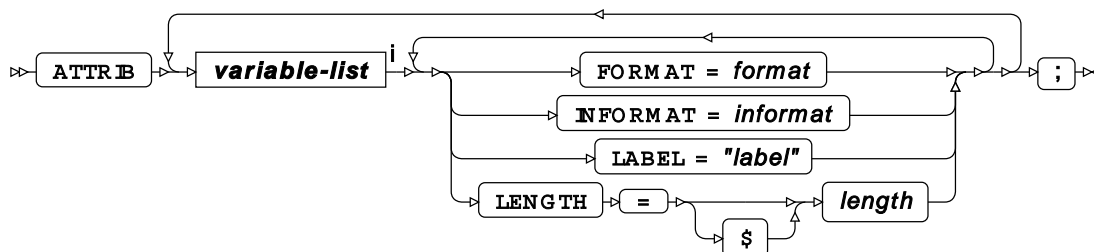
- *PROC KDE* [↗](#) (page 2907)
- *ATTRIB* [↗](#) (page 2907)
- *BIVAR* [↗](#) (page 2908)
- *BY* [↗](#) (page 2908)
- *FORMAT* [↗](#) (page 2908)
- *FREQ* [↗](#) (page 2909)
- *INFORMAT* [↗](#) (page 2909)
- *LABEL* [↗](#) (page 2909)
- *UNIVAR* [↗](#) (page 2909)
- *WEIGHT* [↗](#) (page 2910)
- *WHERE* [↗](#) (page 2910)

PROC KDE



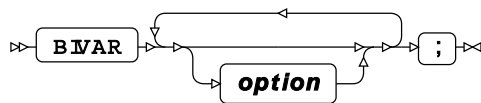
ⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

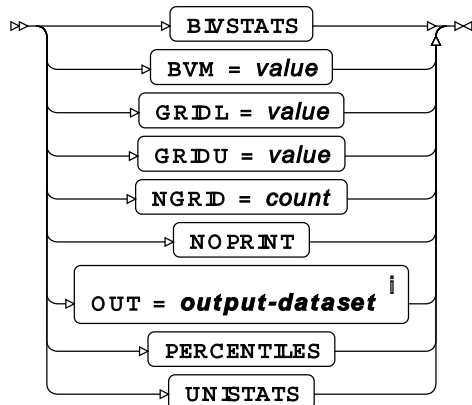


ⁱ See *Variable Lists* [↗](#) (page 32).

BIVAR

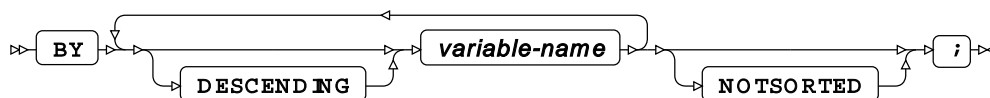


option



ⁱ See [Input dataset](#) (page 16).

BY

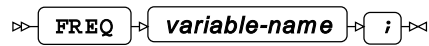


FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

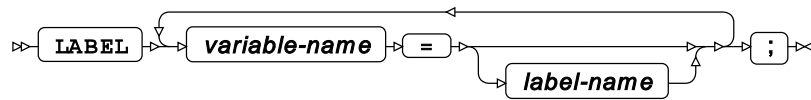


INFORMAT

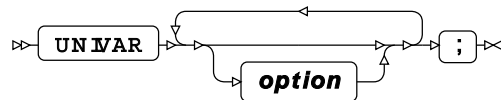


ⁱ See *Variable Lists* [↗](#) (page 32).

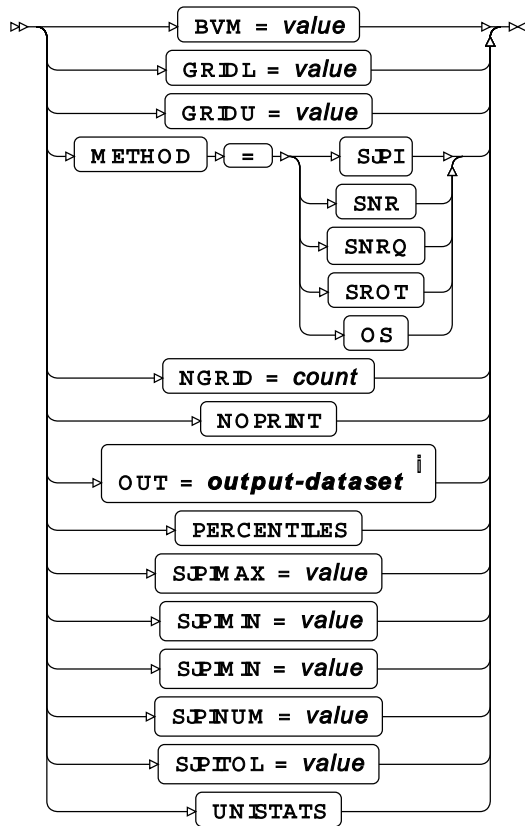
LABEL



UNIVAR



option

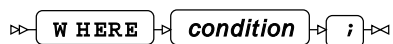


ⁱ See *Input dataset* [↗](#) (page 16).

WEIGHT



WHERE

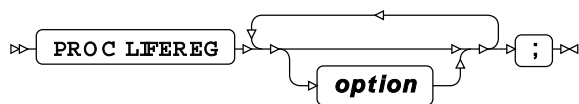


LIFEREG procedure

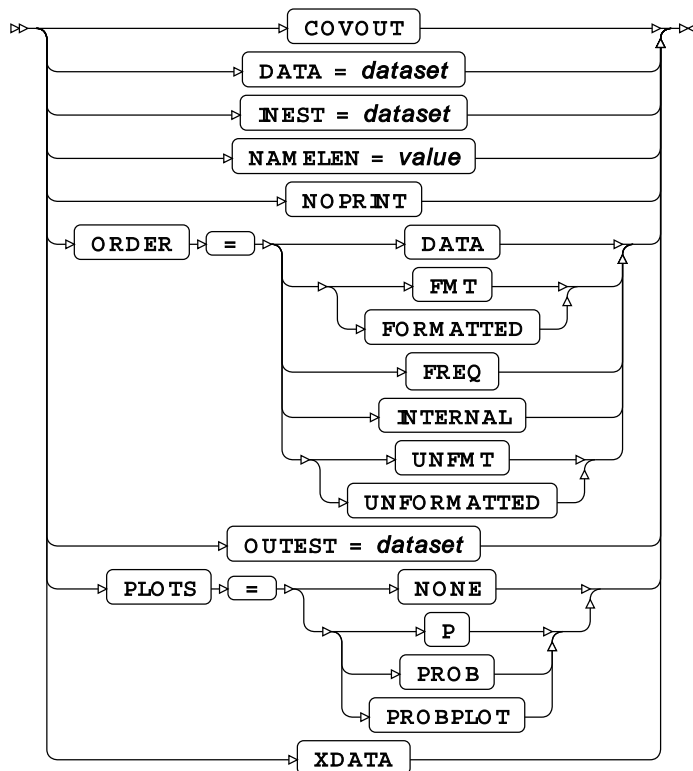
Supported statements

- *PROC LIFEREG* [↗](#) (page 2911)
- *ATTRIB* [↗](#) (page 2912)
- *BY* [↗](#) (page 2912)
- *CLASS* [↗](#) (page 2913)
- *CODE* [↗](#) (page 2913)
- *ESTIMATE* [↗](#) (page 2913)
- *FORMAT* [↗](#) (page 2915)
- *INFORMAT* [↗](#) (page 2915)
- *INSET* [↗](#) (page 2915)
- *LABEL* [↗](#) (page 2916)
- *MODEL* [↗](#) (page 2916)
- *PLOT* [↗](#) (page 2918)
- *OUTPUT* [↗](#) (page 2917)
- *WEIGHT* [↗](#) (page 2920)
- *WHERE* [↗](#) (page 2920)

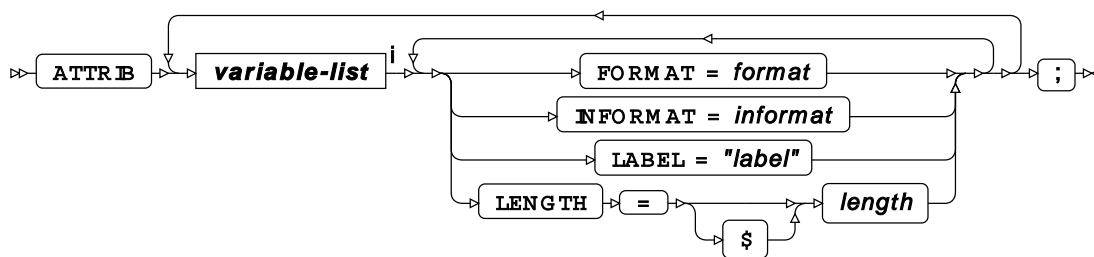
PROC LIFEREG



option

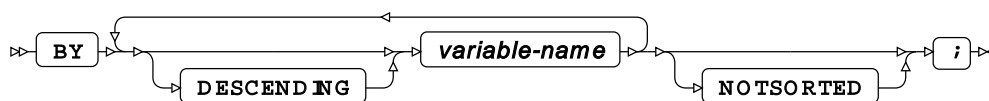


ATTRIB

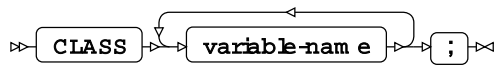


ⁱ See [Variable Lists](#) (page 32).

BY



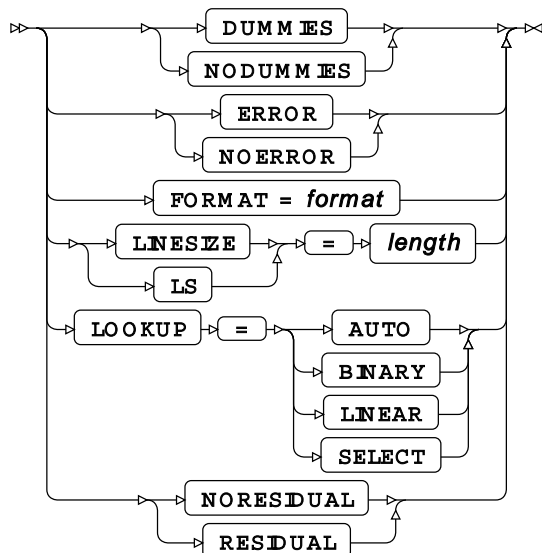
CLASS



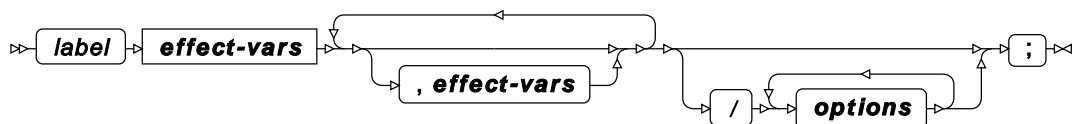
CODE



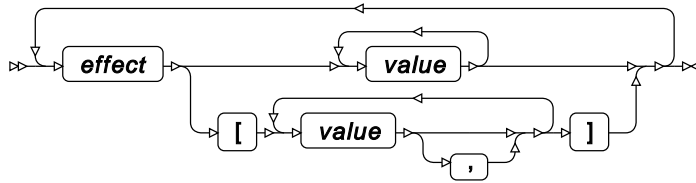
options



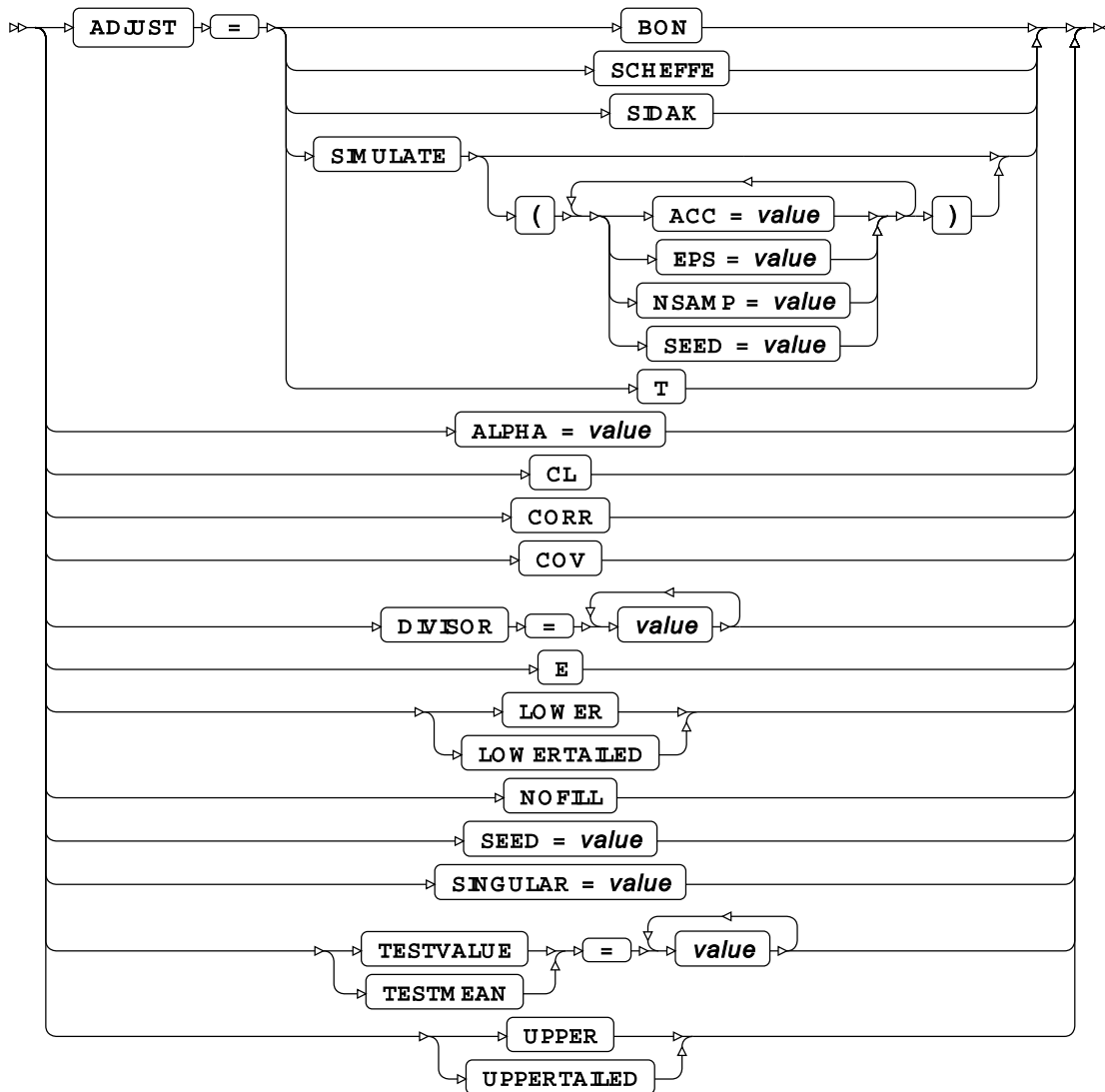
ESTIMATE



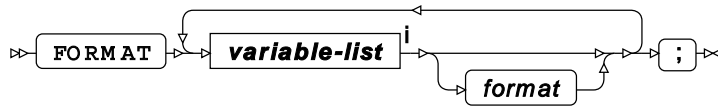
effect-vars



options



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

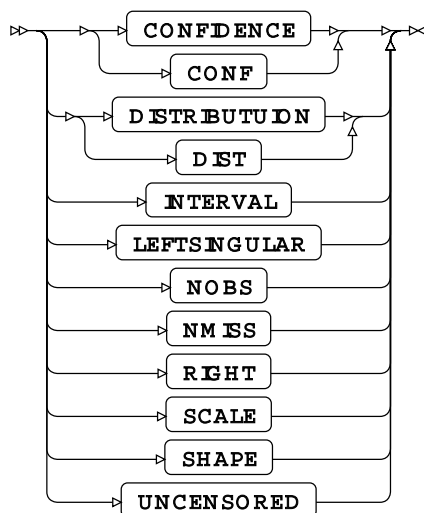


ⁱ See *Variable Lists* [↗](#) (page 32).

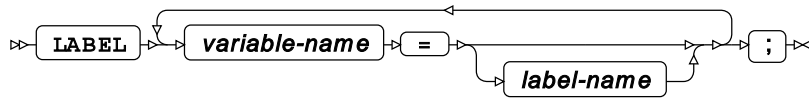
INSET



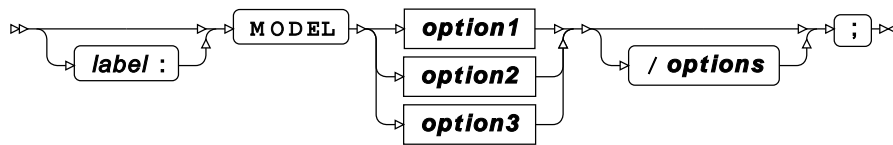
options



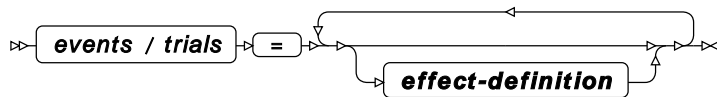
LABEL



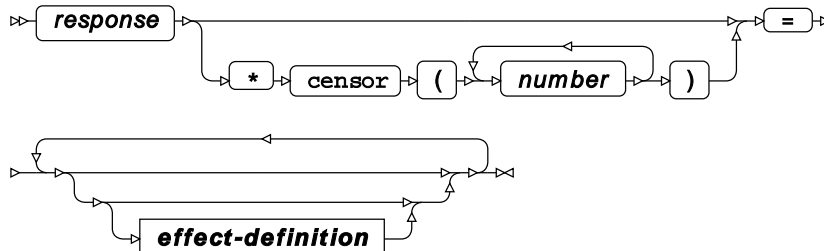
MODEL



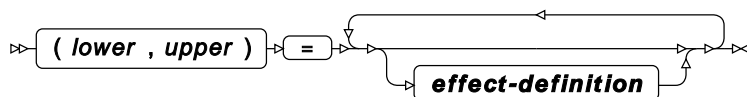
option1



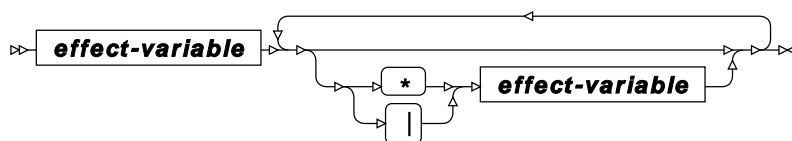
option2



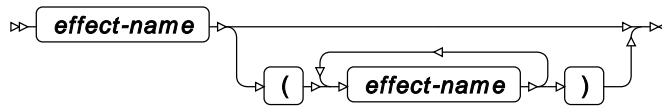
option3



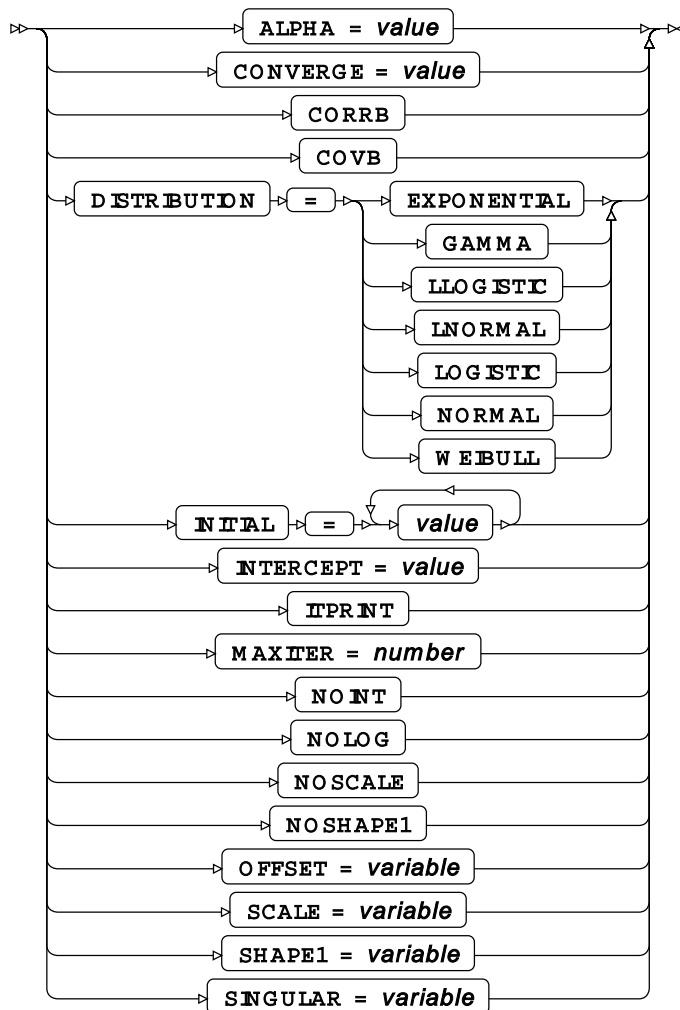
effect-definition



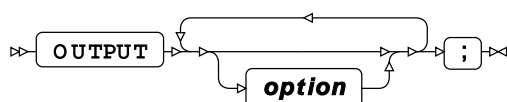
effect-variable



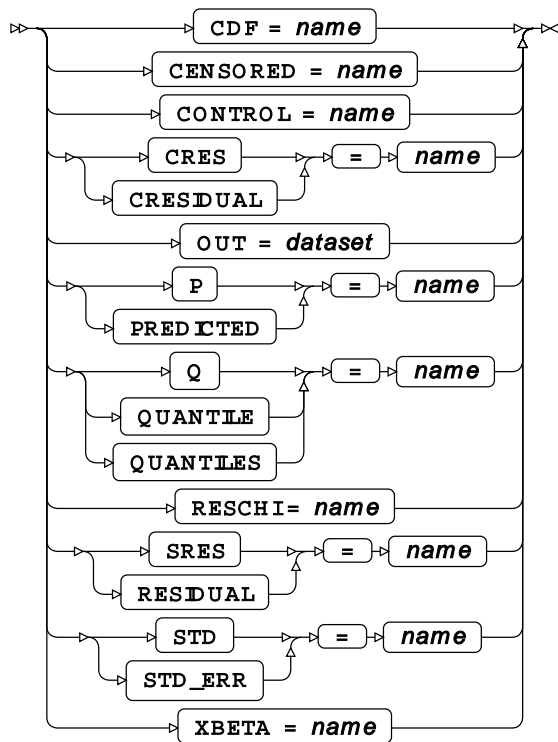
options



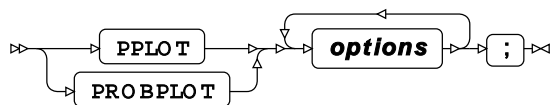
OUTPUT



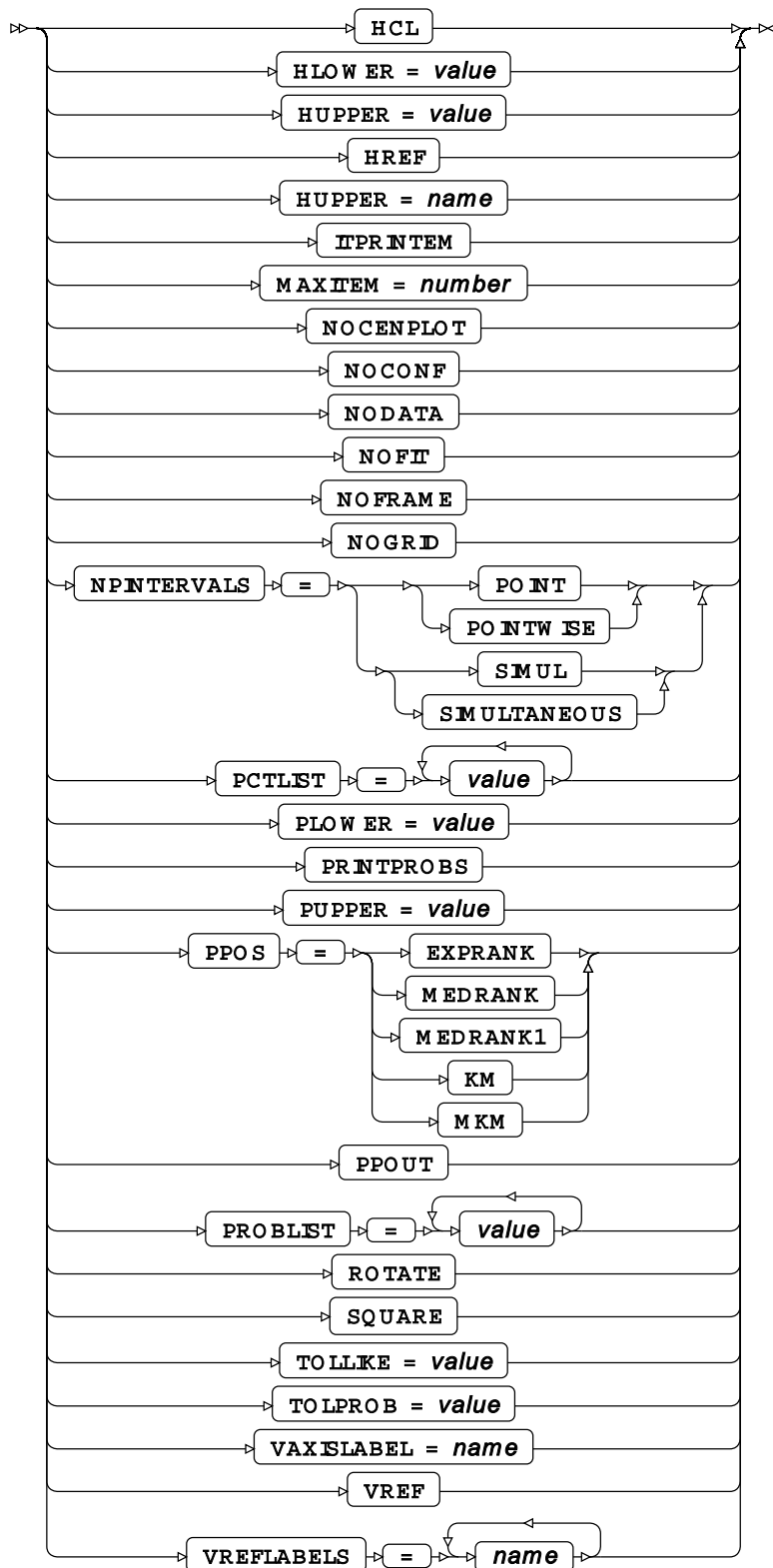
option



PPLOT



options



WEIGHT



WHERE

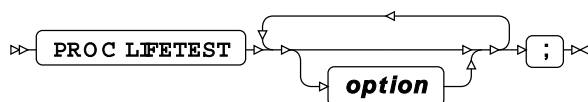


LIFETEST procedure

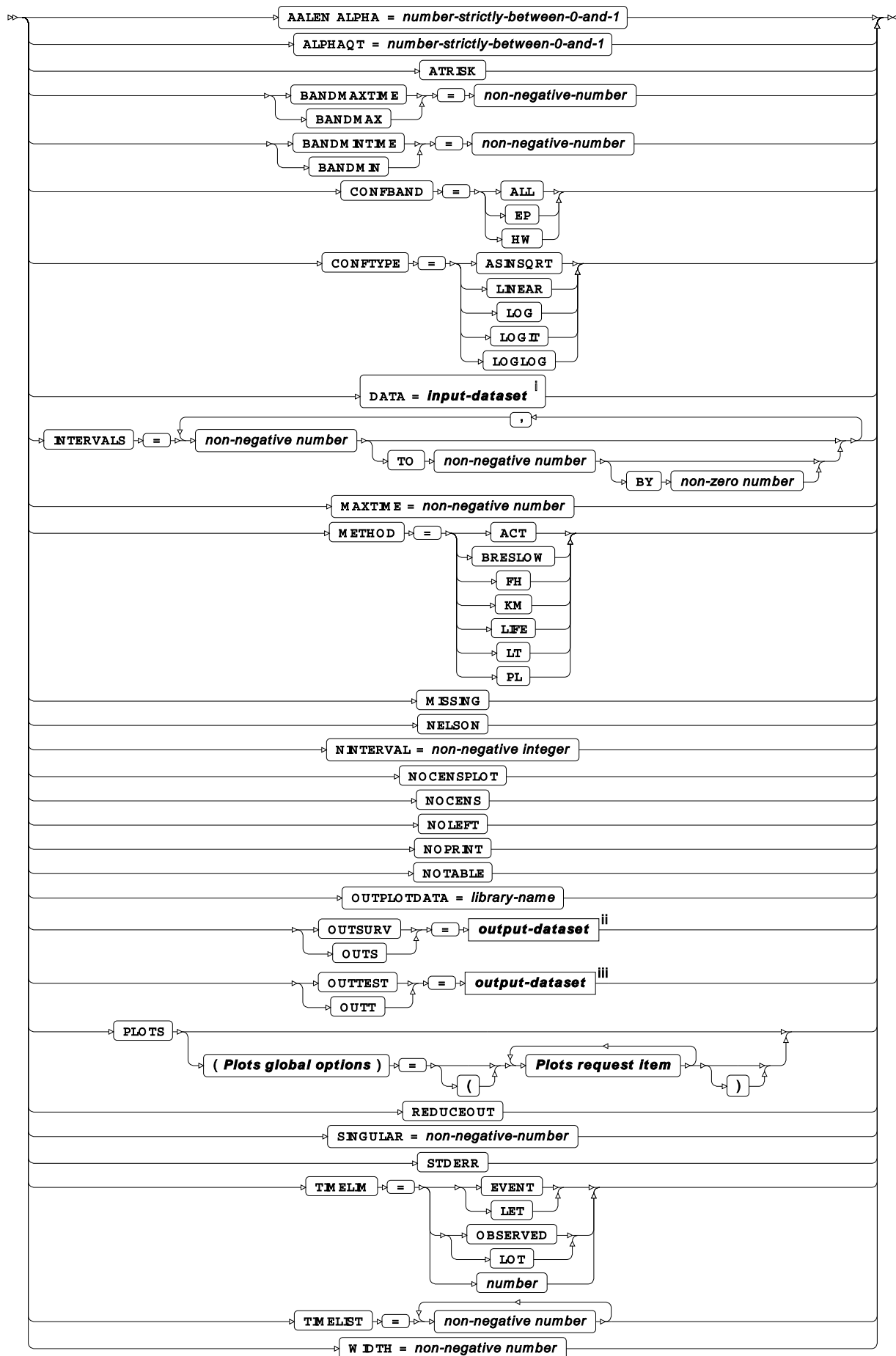
Supported statements

- *PROC LIFETEST* [↗](#) (page 2920)
- *ATTRIB* [↗](#) (page 2923)
- *BY* [↗](#) (page 2924)
- *FORMAT* [↗](#) (page 2924)
- *FREQ* [↗](#) (page 2924)
- *ID* [↗](#) (page 2924)
- *INFORMAT* [↗](#) (page 2924)
- *LABEL* [↗](#) (page 2925)
- *STRATA* [↗](#) (page 2925)
- *TEST* [↗](#) (page 2926)
- *TIME* [↗](#) (page 2926)
- *WHERE* [↗](#) (page 2926)

PROC LIFETEST



option

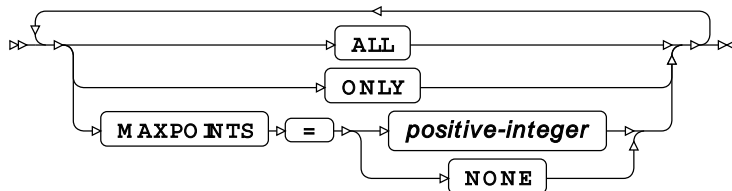


ⁱ See *Input dataset* [↗](#) (page 16).

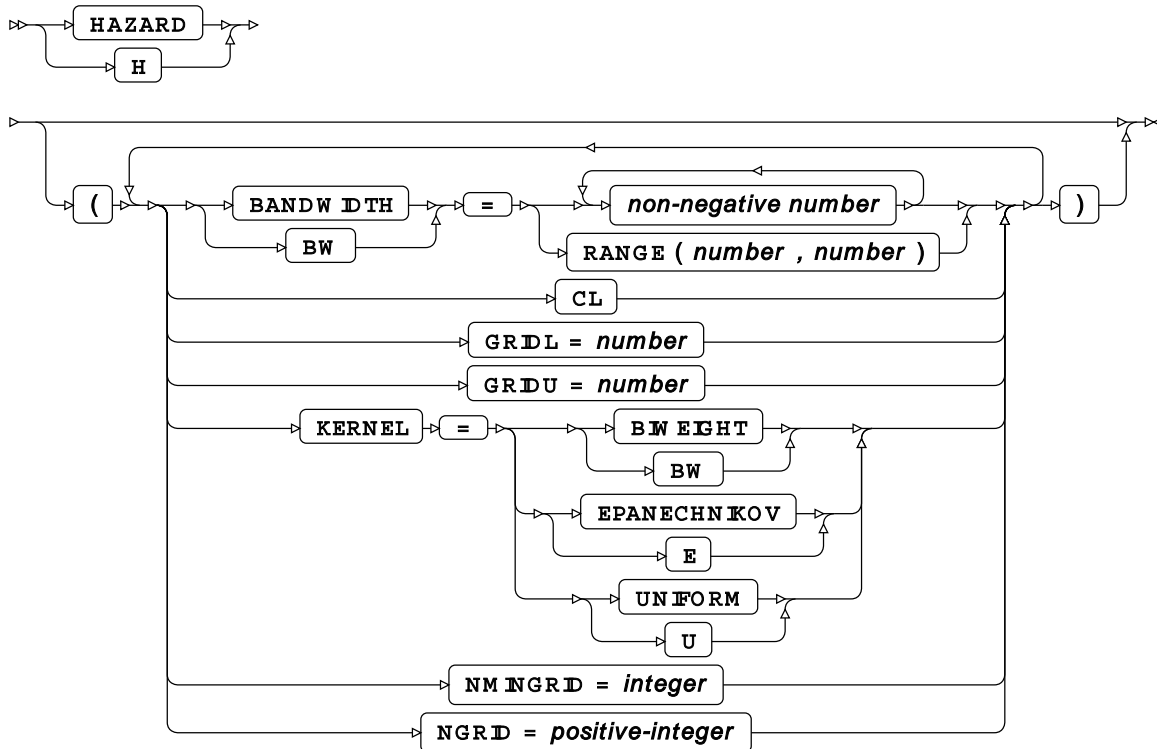
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

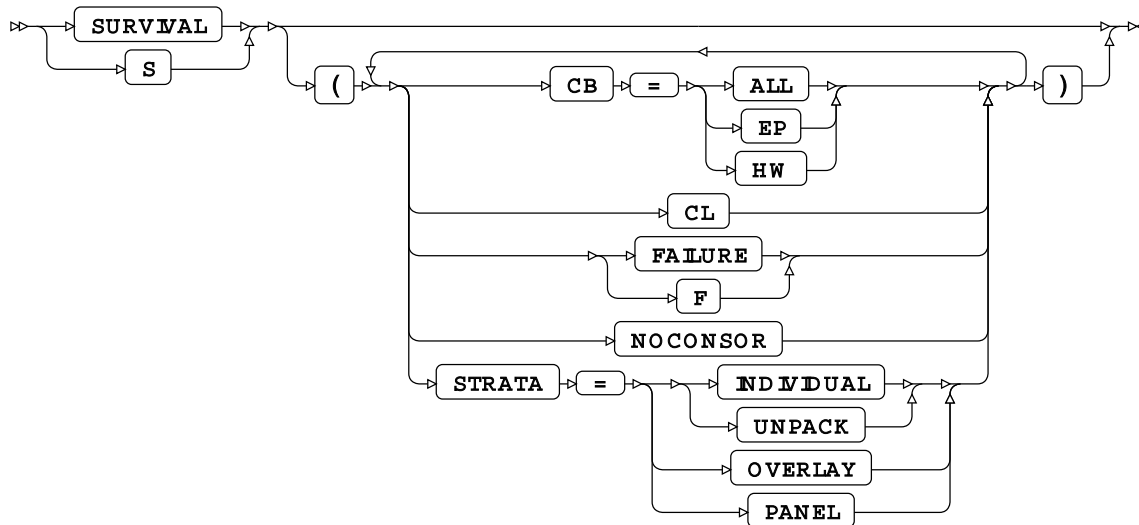
Plots global options



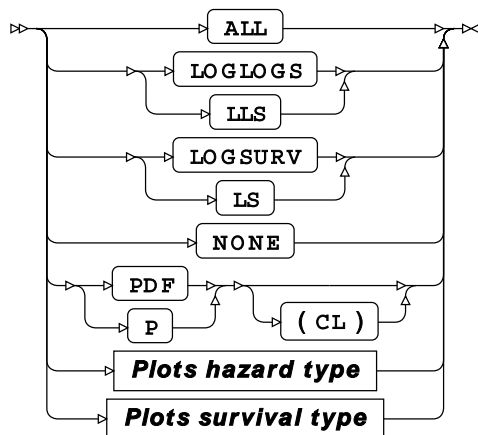
Plots hazard type



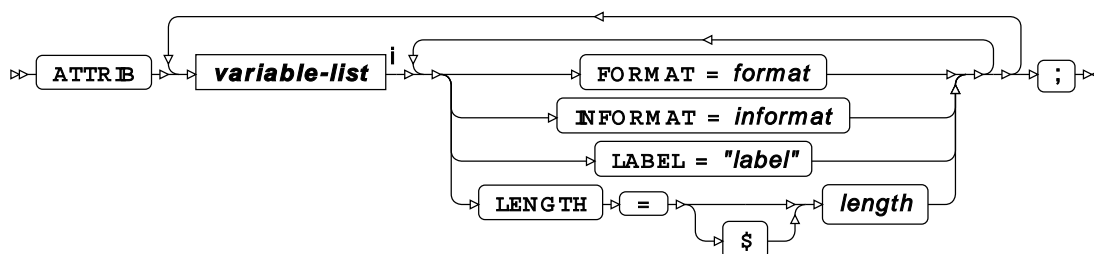
Plots survival type



Plots request item

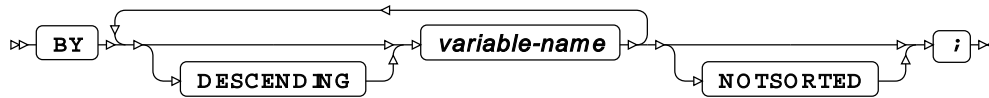


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

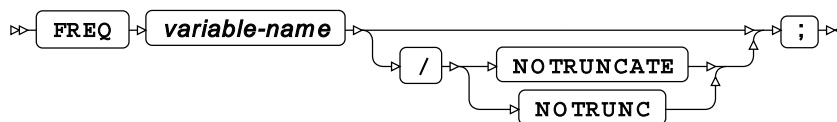


FORMAT

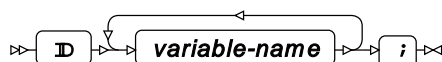


ⁱ See [Variable Lists](#) (page 32).

FREQ



ID

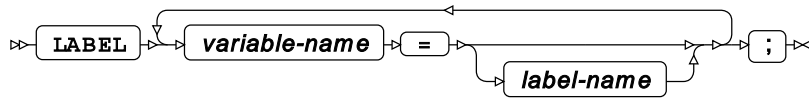


INFORMAT

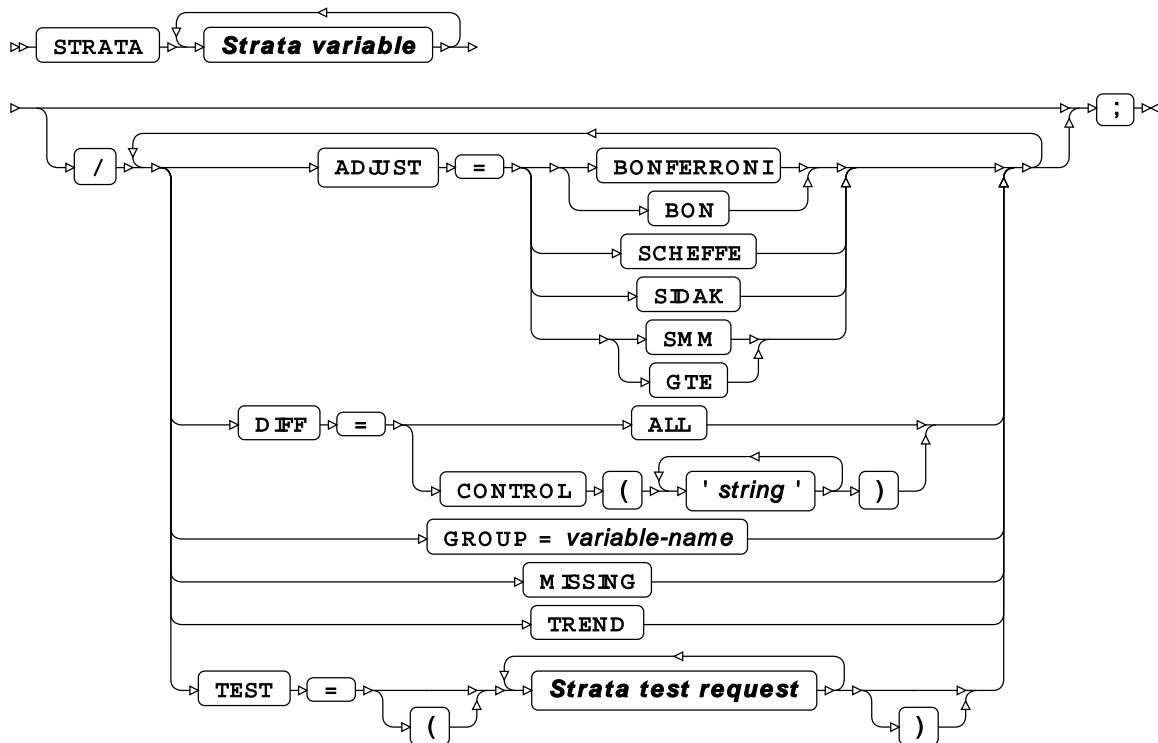


ⁱ See [Variable Lists](#) (page 32).

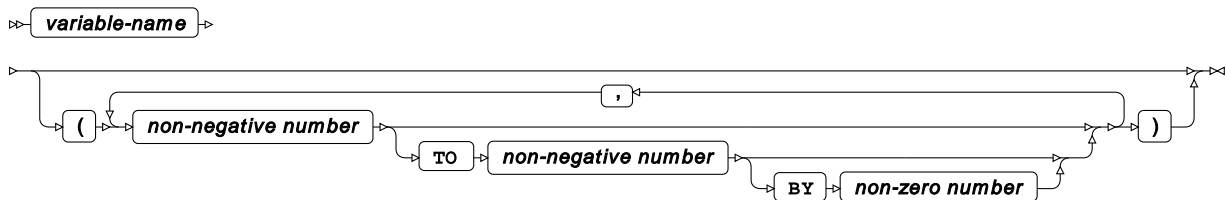
LABEL



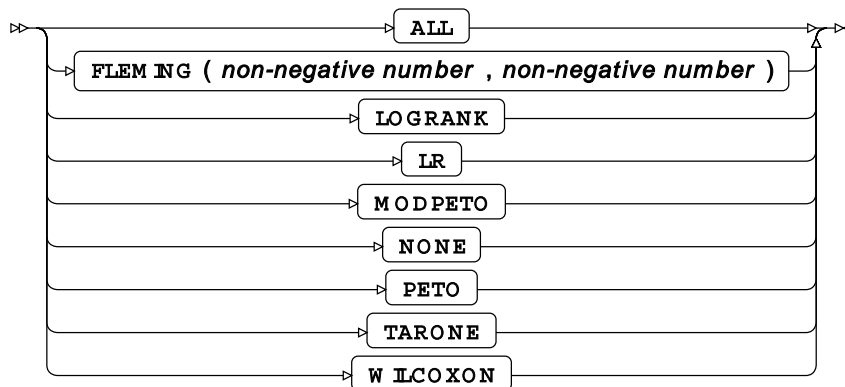
STRATA



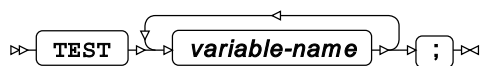
Strata variable



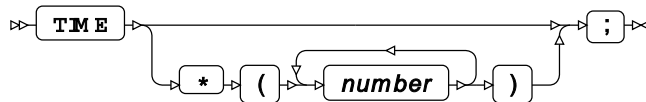
Strata test request



TEST



TIME



WHERE



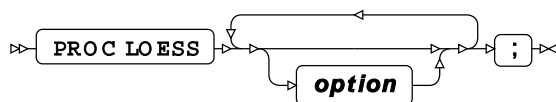
LOESS procedure

Supported statements

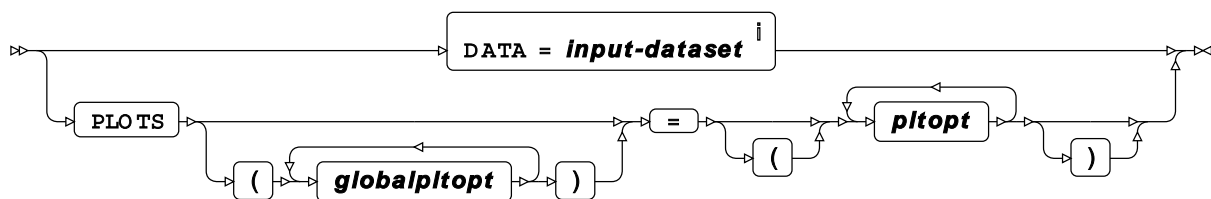
- *PROC LOESS* [↗](#) (page 2927)
- *ATTRIB* [↗](#) (page 2929)
- *BY* [↗](#) (page 2929)
- *FORMAT* [↗](#) (page 2929)

- [ID](#) (page 2929)
- [INFORMAT](#) (page 2929)
- [LABEL](#) (page 2930)
- [MODEL](#) (page 2931)
- [OUTPUT](#) (page 2932)
- [SCORE](#) (page 2932)
- [WEIGHT](#) (page 2933)
- [WHERE](#) (page 2933)

PROC LOESS

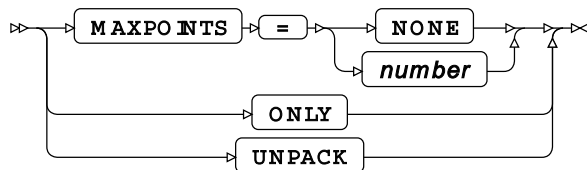


option

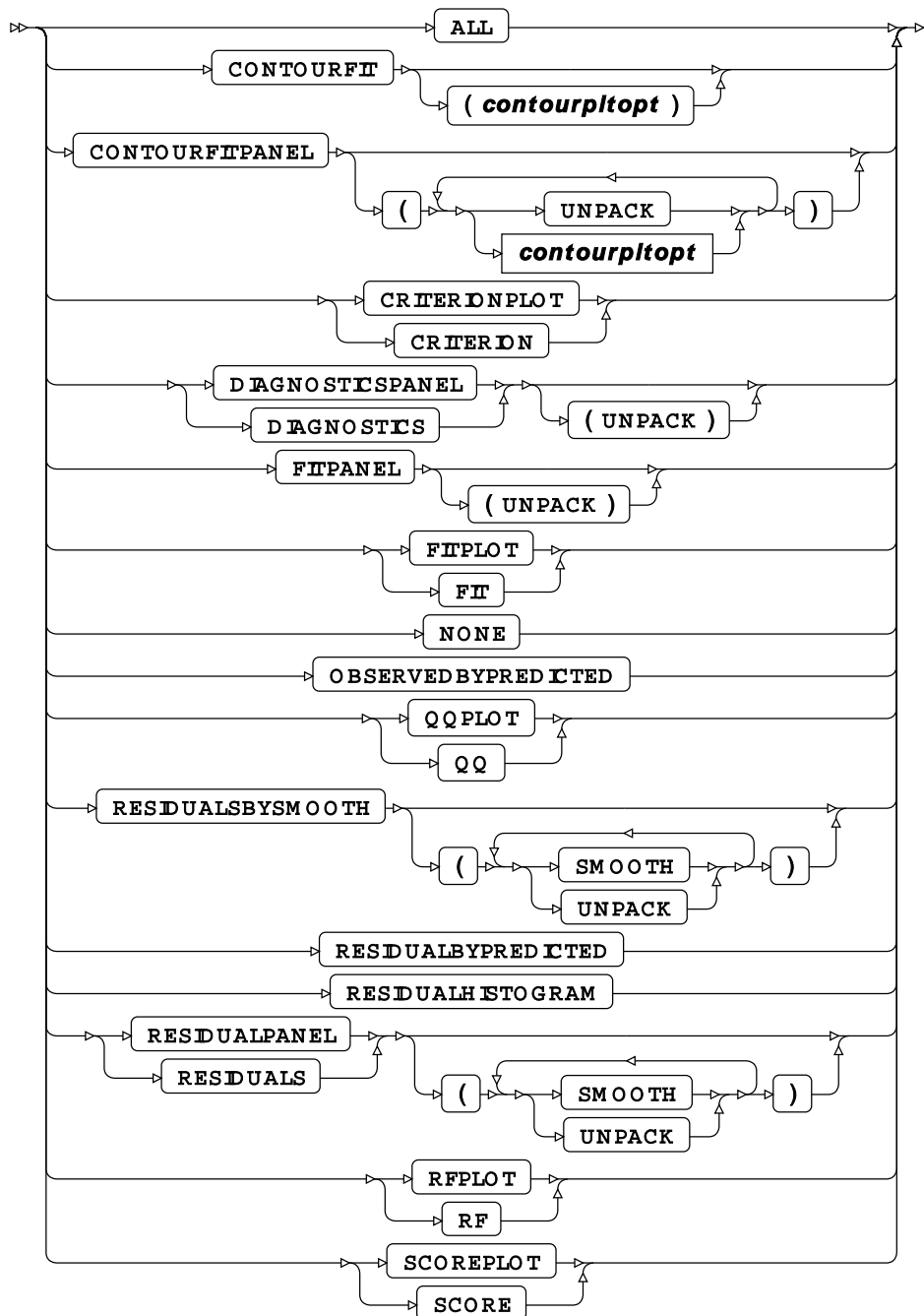


ⁱ See [Input dataset](#) (page 16).

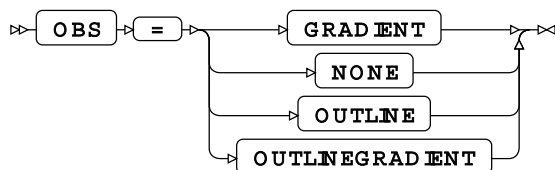
globalpltopt



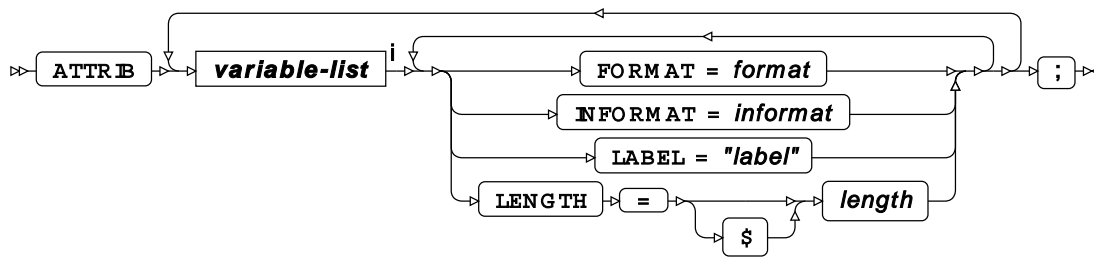
pltopt



contourpltopt

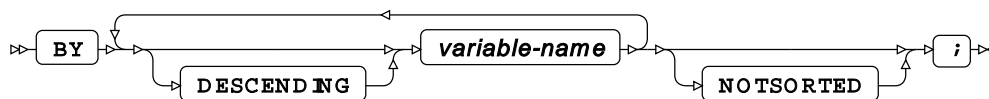


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

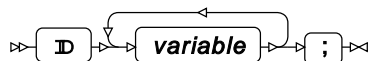


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

ID

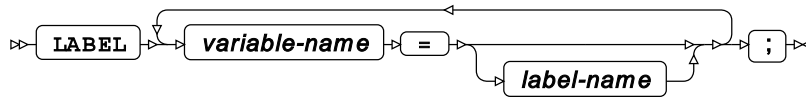


INFORMAT

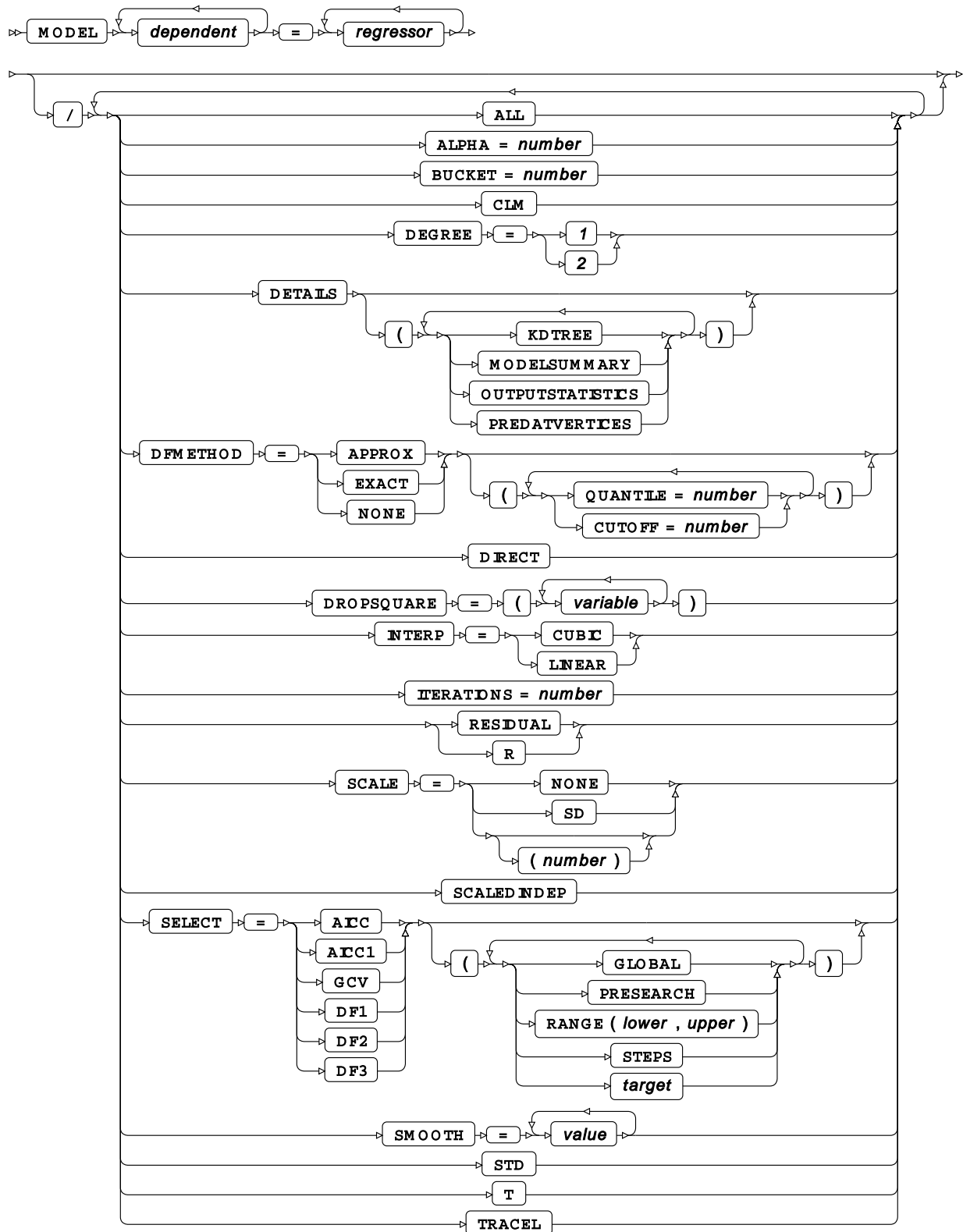


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL

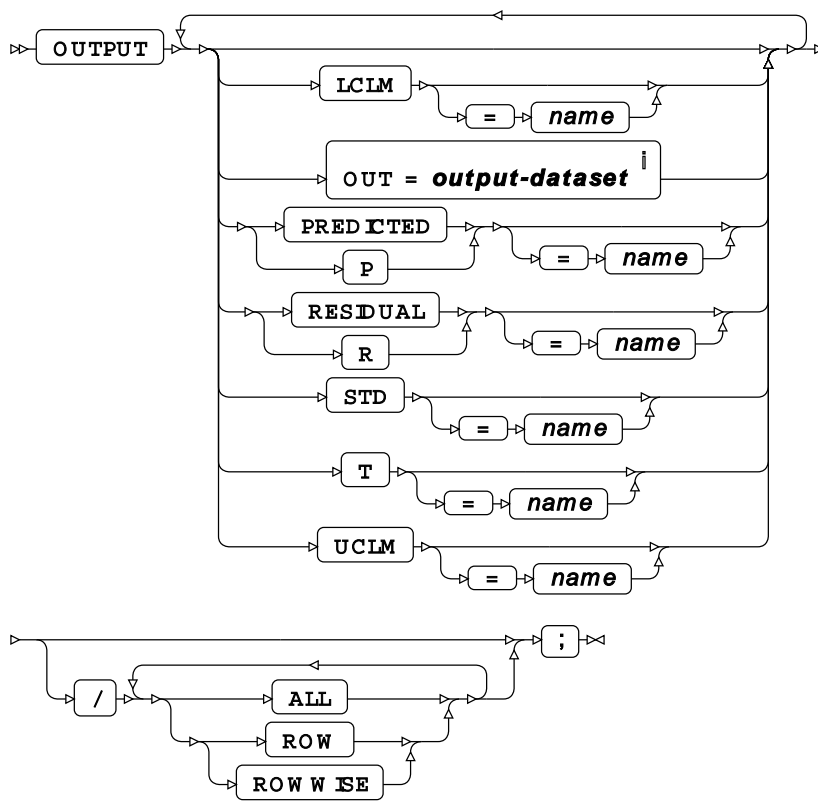


MODEL



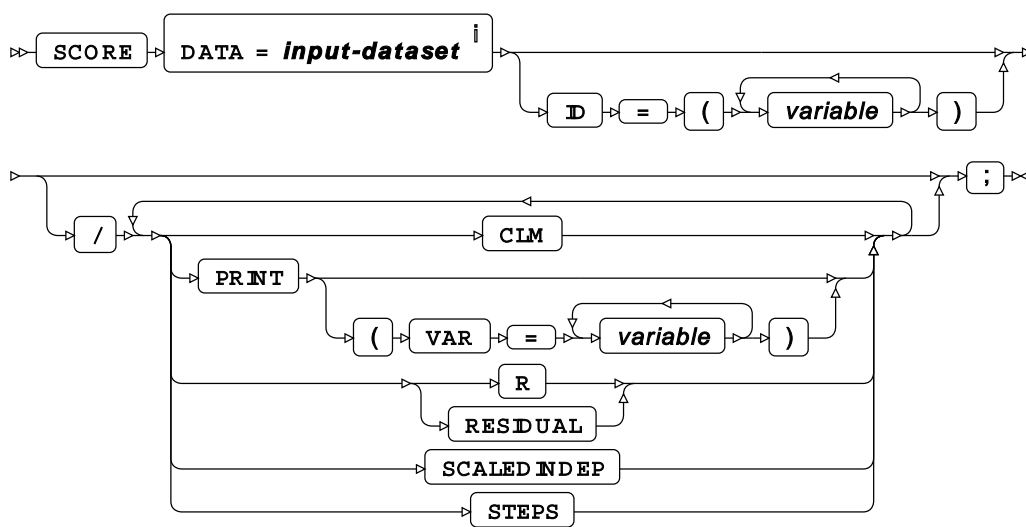
;

OUTPUT



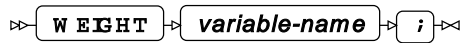
ⁱ See *Output dataset* [\[link\]](#) (page 16).

SCORE

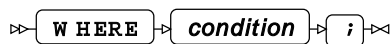


ⁱ See *Input dataset* [↗](#) (page 16).

WEIGHT



WHERE

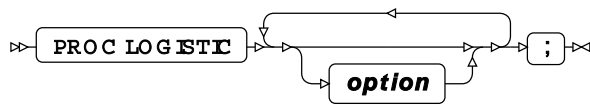


LOGISTIC procedure

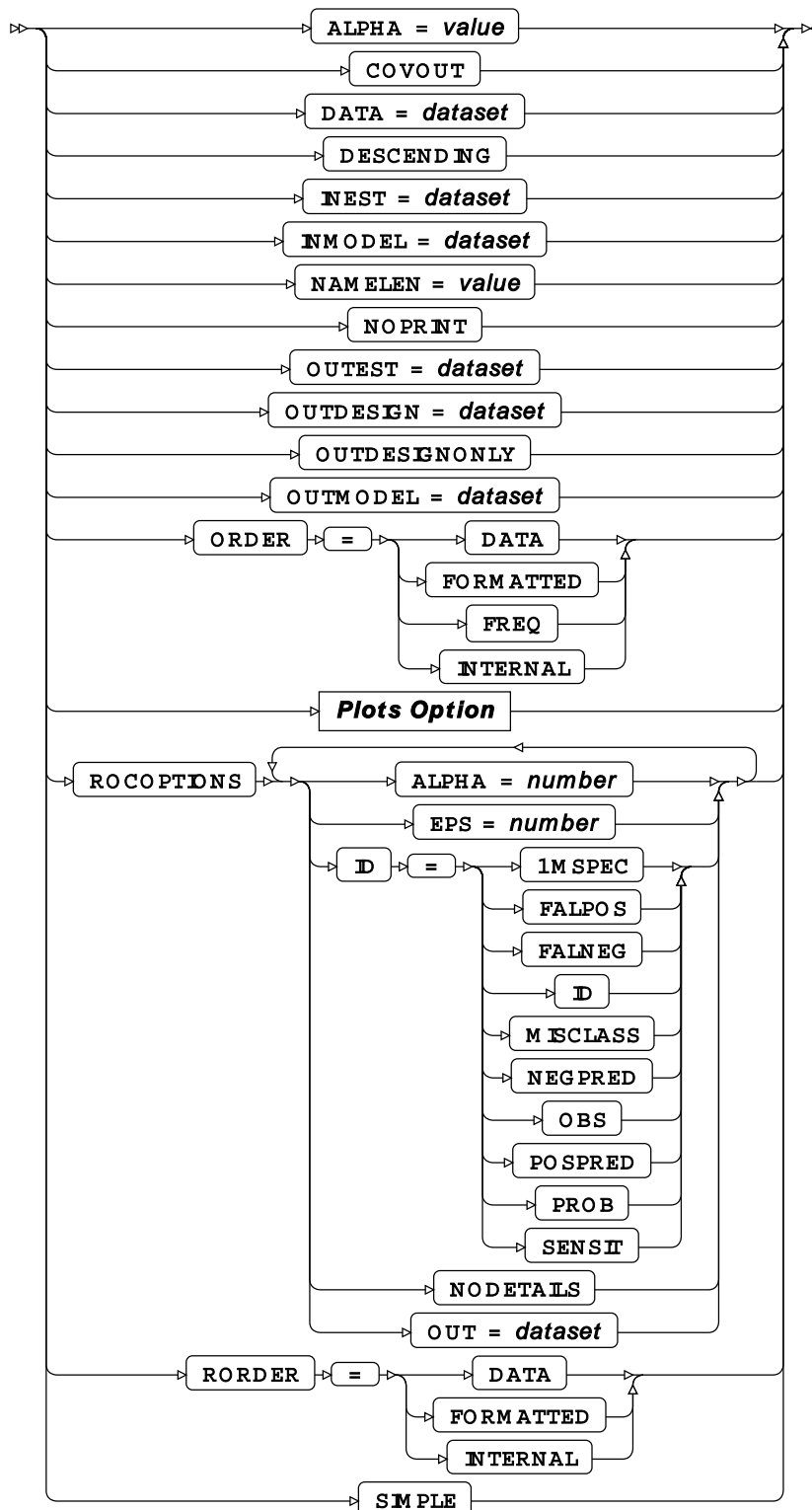
Supported statements

- *PROC LOGISTIC* [↗](#) (page 2934)
- *ATTRIB* [↗](#) (page 2936)
- *BY* [↗](#) (page 2937)
- *CLASS* [↗](#) (page 2937)
- *CODE* [↗](#) (page 2938)
- *CONTRAST* [↗](#) (page 2938)
- *ESTIMATE* [↗](#) (page 2939)
- *FORMAT* [↗](#) (page 2940)
- *INFORMAT* [↗](#) (page 2941)
- *LABEL* [↗](#) (page 2941)
- *MODEL* [↗](#) (page 2941)
- *FREQ* [↗](#) (page 2944)
- *OUTPUT* [↗](#) (page 2944)
- *ROC* [↗](#) (page 2945)
- *ROCCONTRAST* [↗](#) (page 2946)
- *SCORE* [↗](#) (page 2947)
- *TEST* [↗](#) (page 2947)
- *WEIGHT* [↗](#) (page 2947)
- *WHERE* [↗](#) (page 2947)

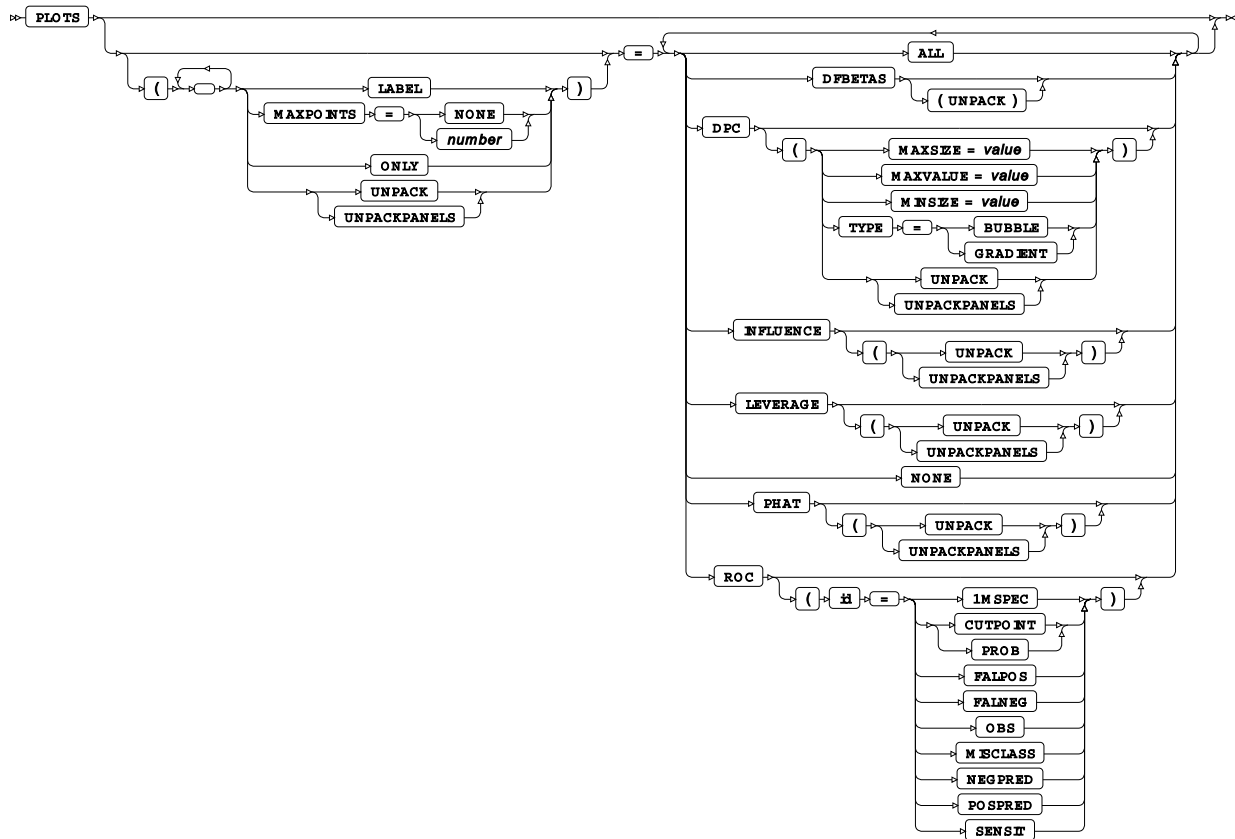
PROC LOGISTIC



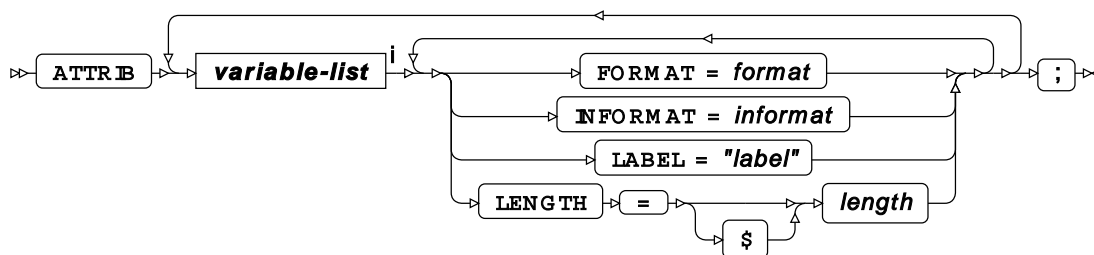
option



Plots Option

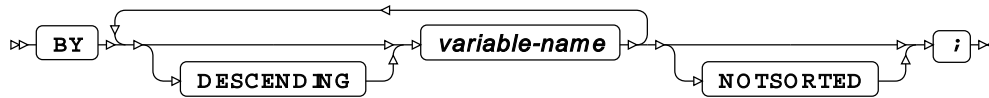


ATTRIB

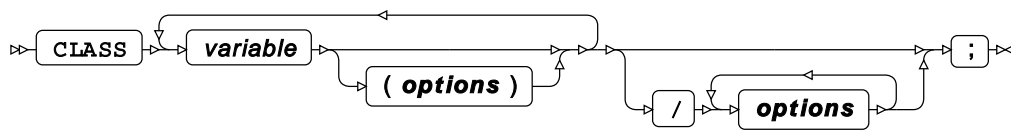


ⁱ See *Variable Lists* [↗](#) (page 32).

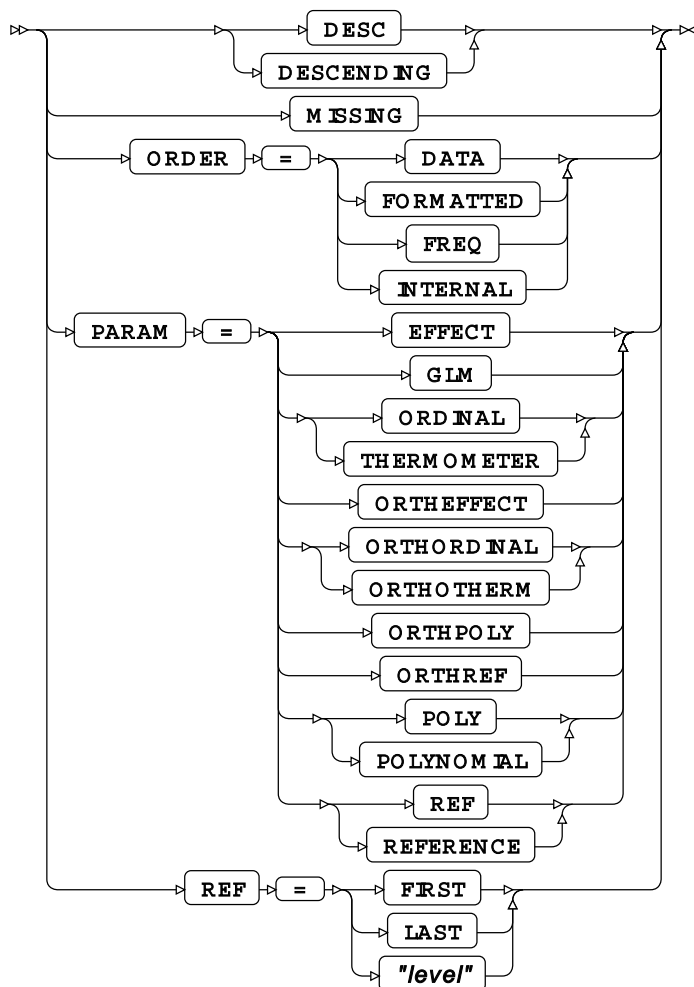
BY



CLASS



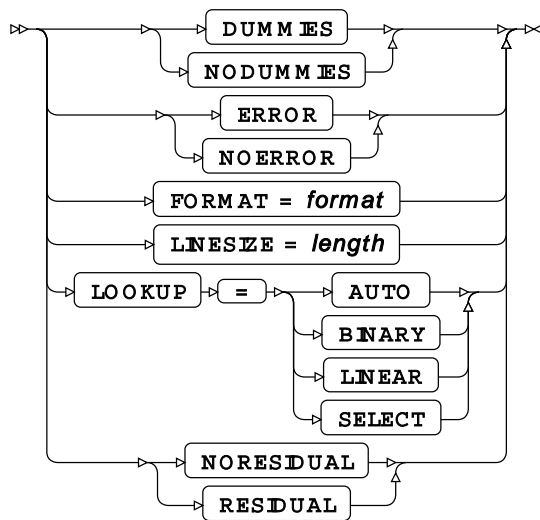
options



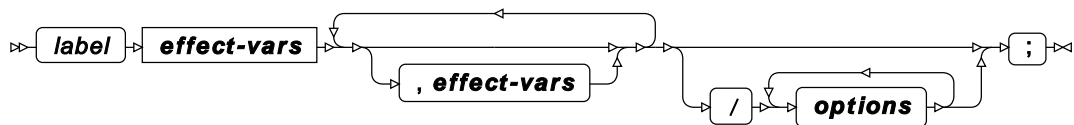
CODE



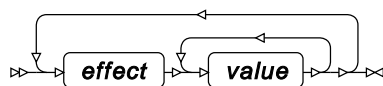
options



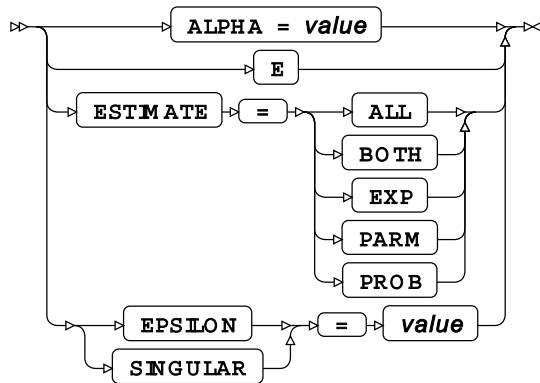
CONTRAST



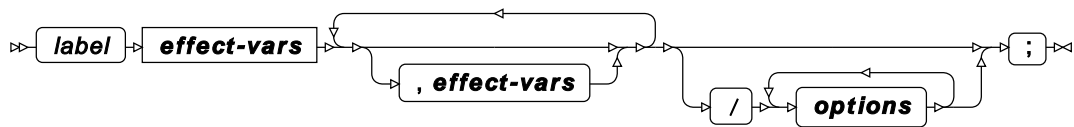
effect-vars



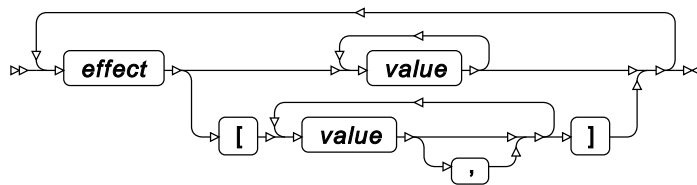
options



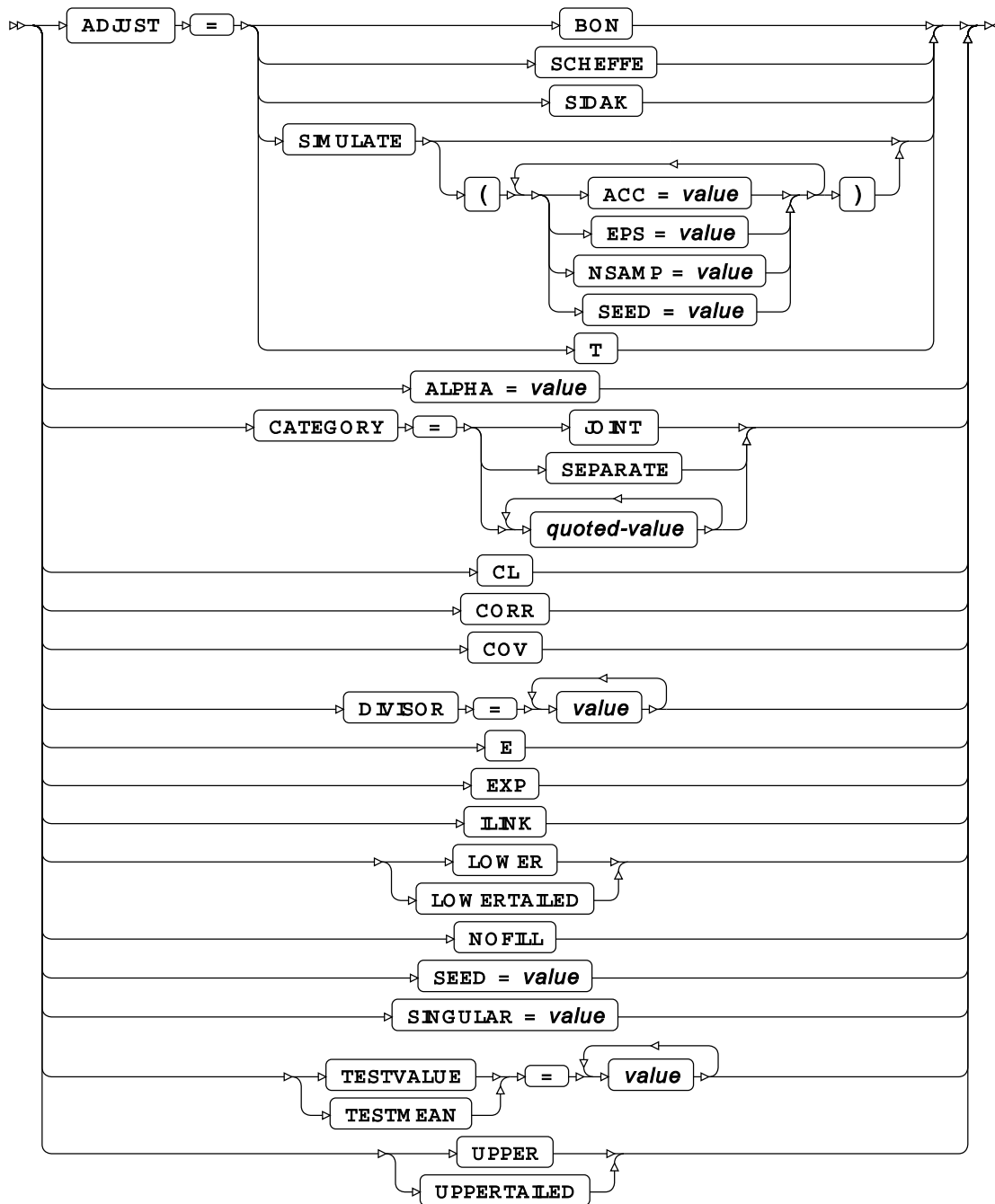
ESTIMATE



effect-vars



options



FORMAT



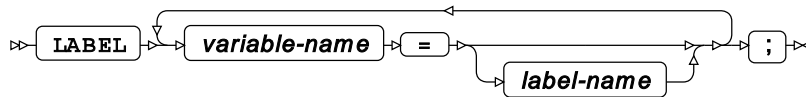
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

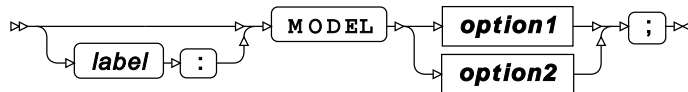


ⁱ See *Variable Lists* [↗](#) (page 32).

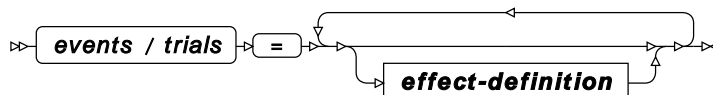
LABEL



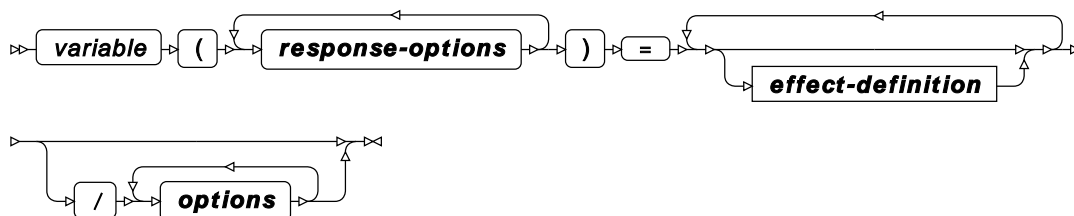
MODEL



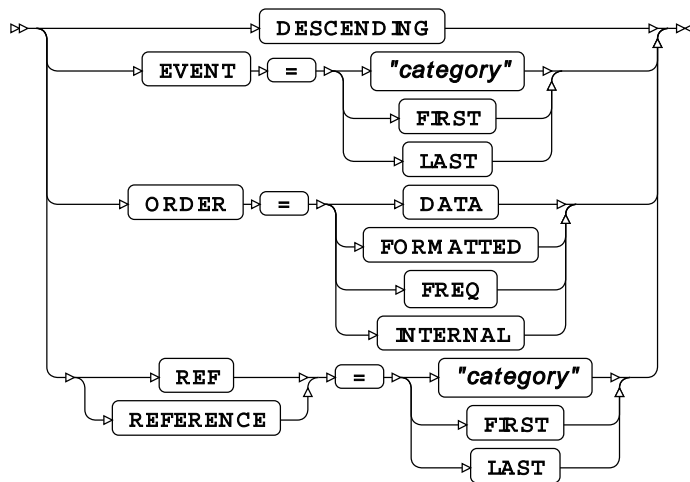
option1



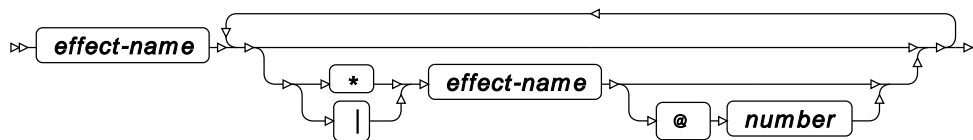
option2



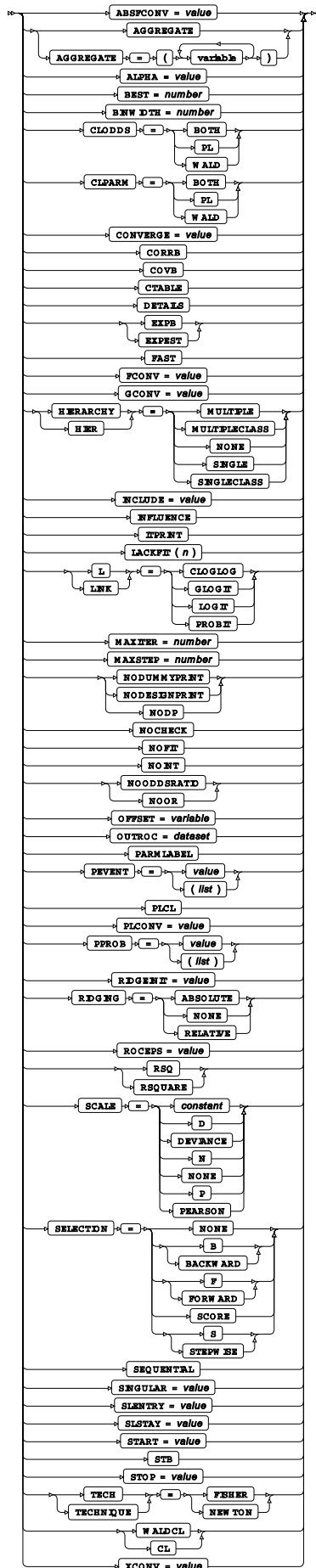
response-options



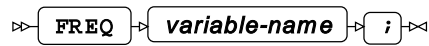
effect-definition



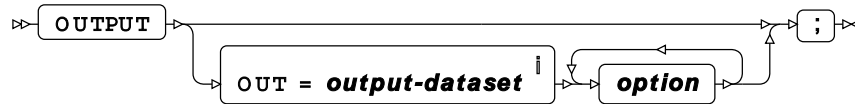
options



FREQ

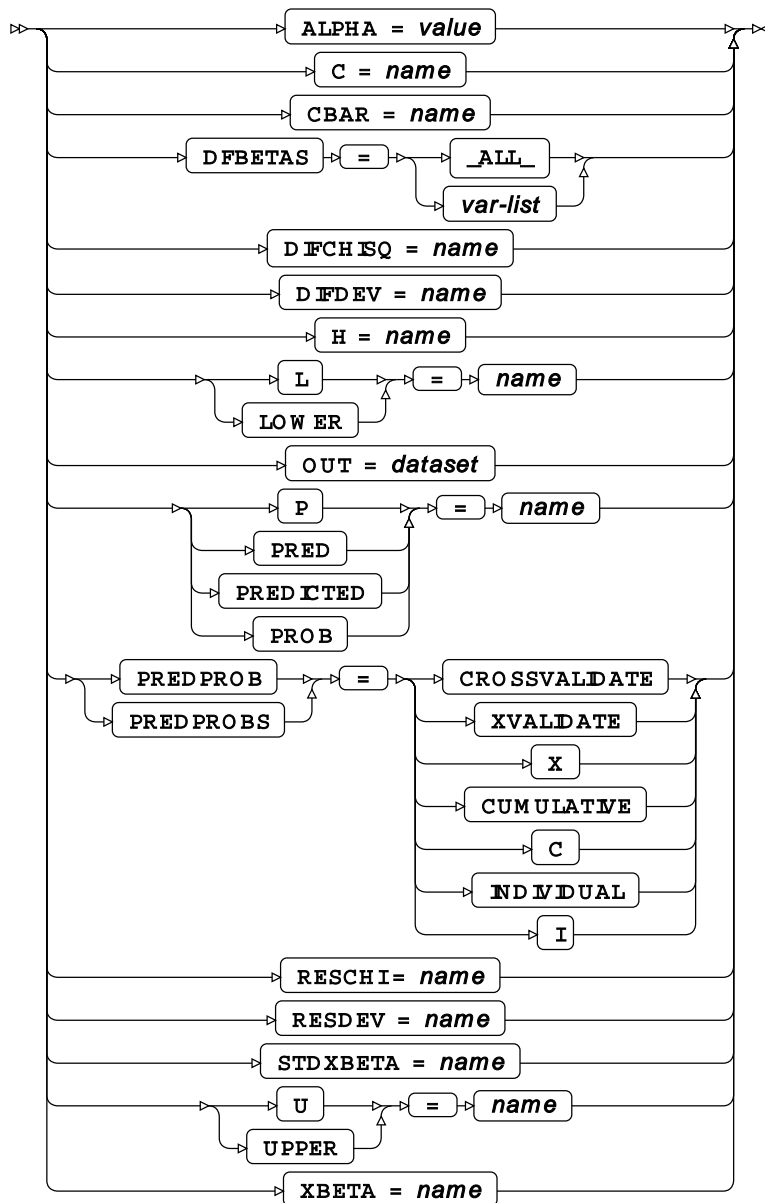


OUTPUT

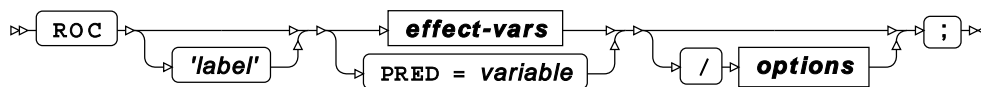


ⁱ See *Output dataset* [↗](#) (page 16).

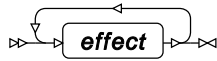
option



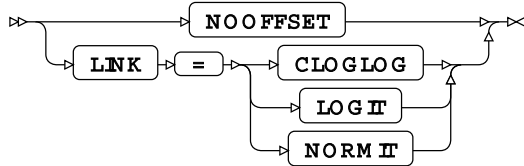
ROC



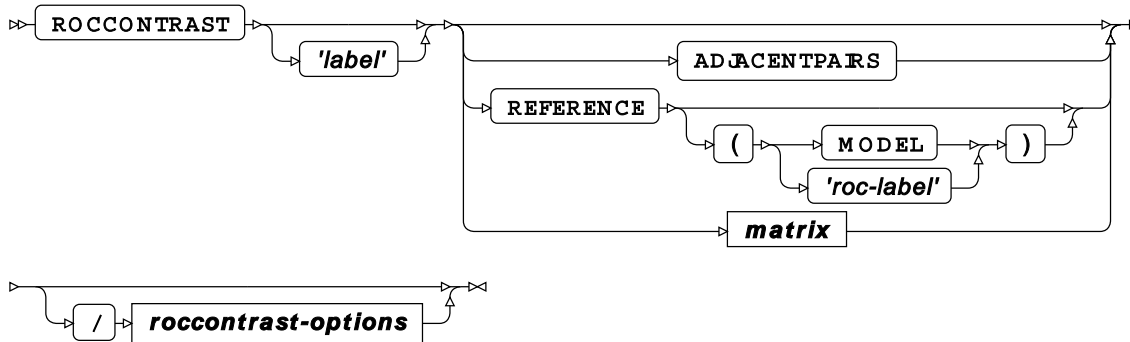
effect-vars



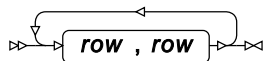
options



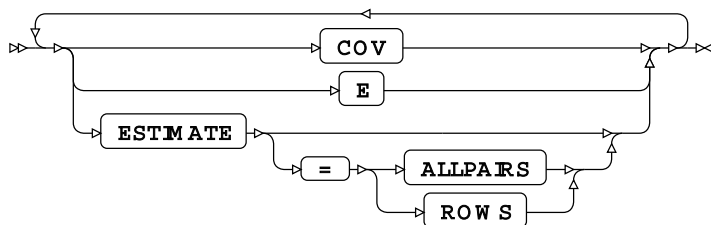
ROCONTRAST



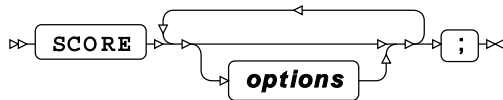
matrix



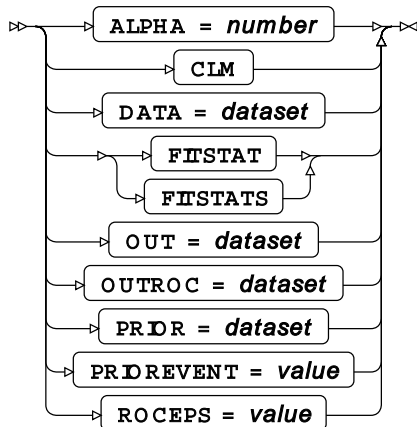
rocontrast-options



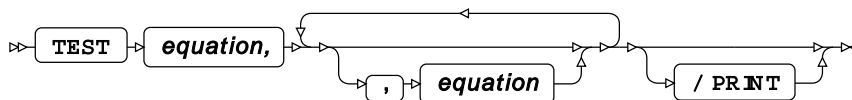
SCORE



options



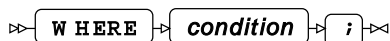
TEST



WEIGHT



WHERE

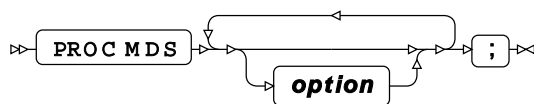


MDS procedure

Supported statements

- *PROC MDS* [↗](#) (page 2948)
- *ATTRIB* [↗](#) (page 2950)
- *BY* [↗](#) (page 2950)
- *FORMAT* [↗](#) (page 2951)
- *ID* [↗](#) (page 2951)
- *INFORMAT* [↗](#) (page 2951)
- *INVAR* [↗](#) (page 2951)
- *LABEL* [↗](#) (page 2951)
- *MATRIX* [↗](#) (page 2951)
- *VAR* [↗](#) (page 2952)
- *WEIGHT* [↗](#) (page 2952)
- *WHERE* [↗](#) (page 2952)

PROC MDS

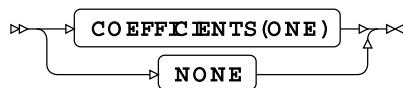


- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Output dataset* [↗](#) (page 16).
- ^{iv} See *Output dataset* [↗](#) (page 16).
- ^v See *Output dataset* [↗](#) (page 16).

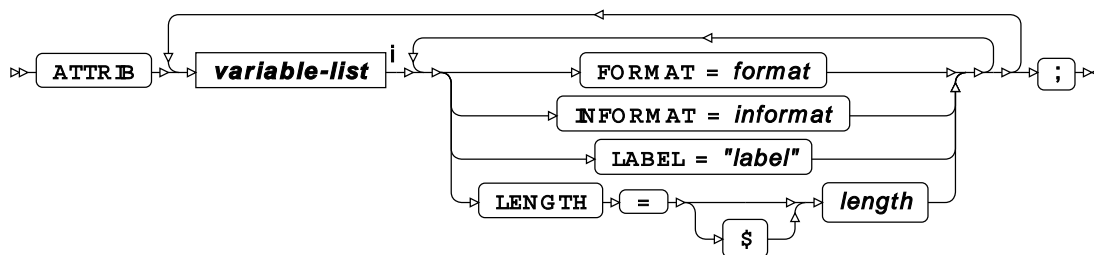
GlobalPlotOptions



PlotRequest

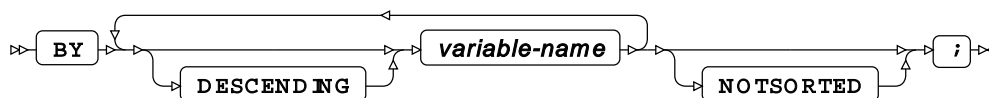


ATTRIB



- ⁱ See *Variable Lists* [↗](#) (page 32).

BY

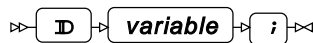


FORMAT

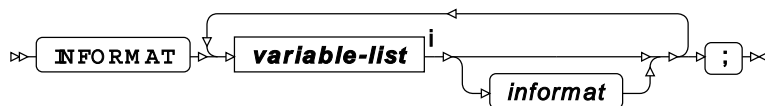


ⁱ See *Variable Lists* [↗](#) (page 32).

ID

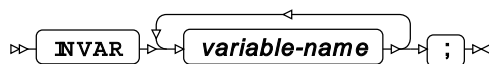


INFORMAT

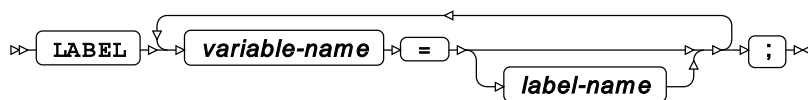


ⁱ See *Variable Lists* [↗](#) (page 32).

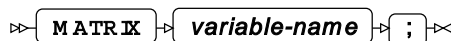
INVAR



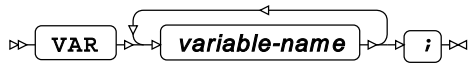
LABEL



MATRIX



VAR



WEIGHT



WHERE

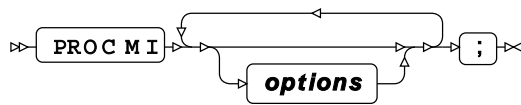


MI procedure

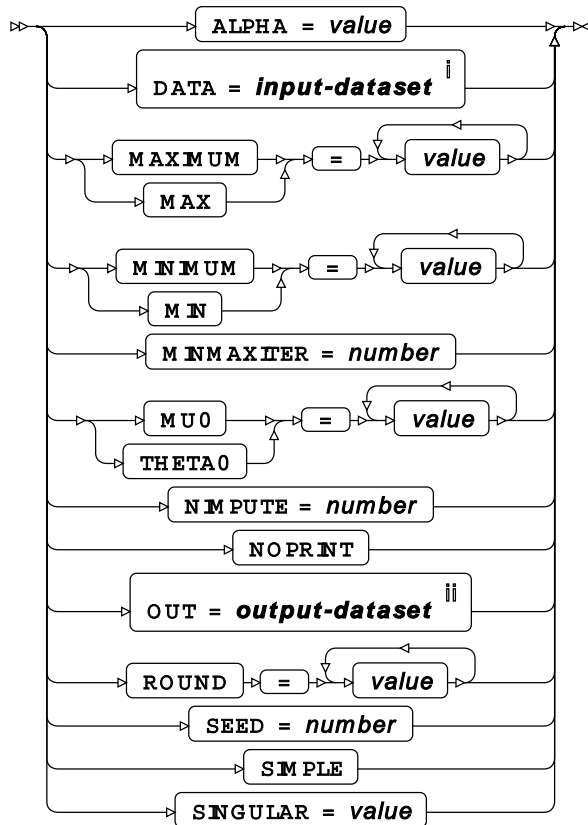
Supported statements

- *PROC MI* [↗](#) (page 2953)
- *ATTRIB* [↗](#) (page 2954)
- *BY* [↗](#) (page 2954)
- *CLASS* [↗](#) (page 2954)
- *EM* [↗](#) (page 2954)
- *FCS* [↗](#) (page 2955)
- *FORMAT* [↗](#) (page 2956)
- *FREQ* [↗](#) (page 2957)
- *INFORMAT* [↗](#) (page 2957)
- *LABEL* [↗](#) (page 2957)
- *MCMC* [↗](#) (page 2957)
- *MONOTONE* [↗](#) (page 2959)
- *TRANSFORM* [↗](#) (page 2961)
- *VAR* [↗](#) (page 2961)
- *WHERE* [↗](#) (page 2961)

PROC MI



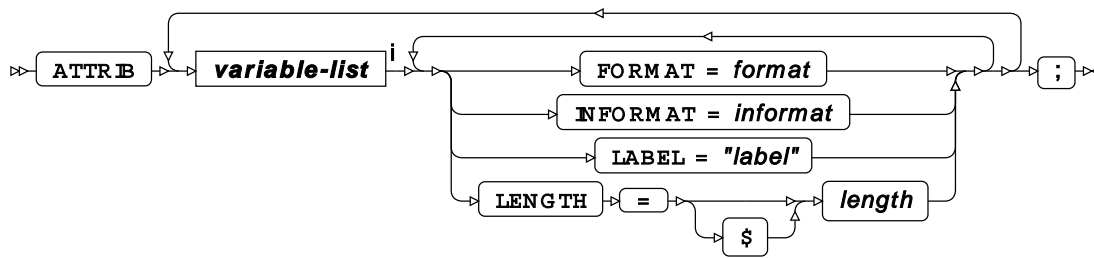
options



ⁱ See *Input dataset* [↗](#) (page 16).

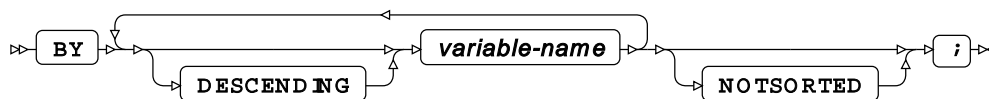
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

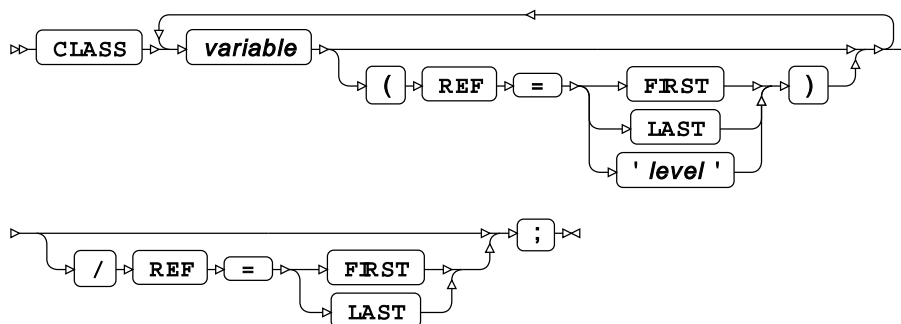


ⁱ See *Variable Lists* [↗](#) (page 32).

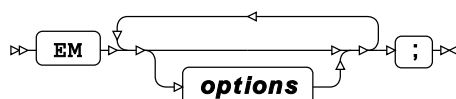
BY



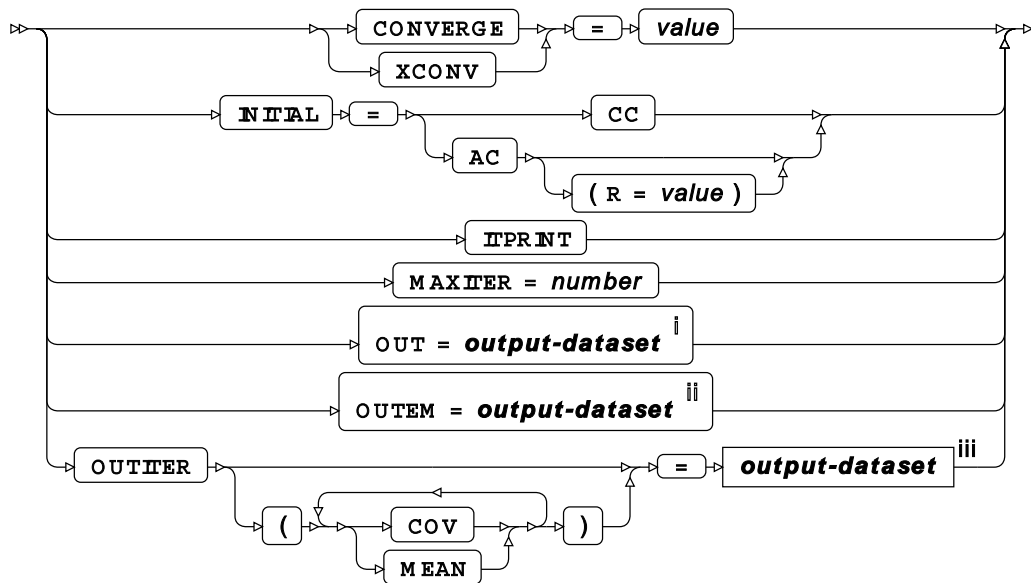
CLASS



EM



options

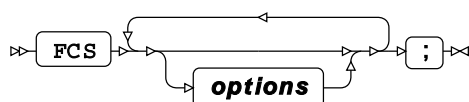


ⁱ See *Input dataset* [↗](#) (page 16).

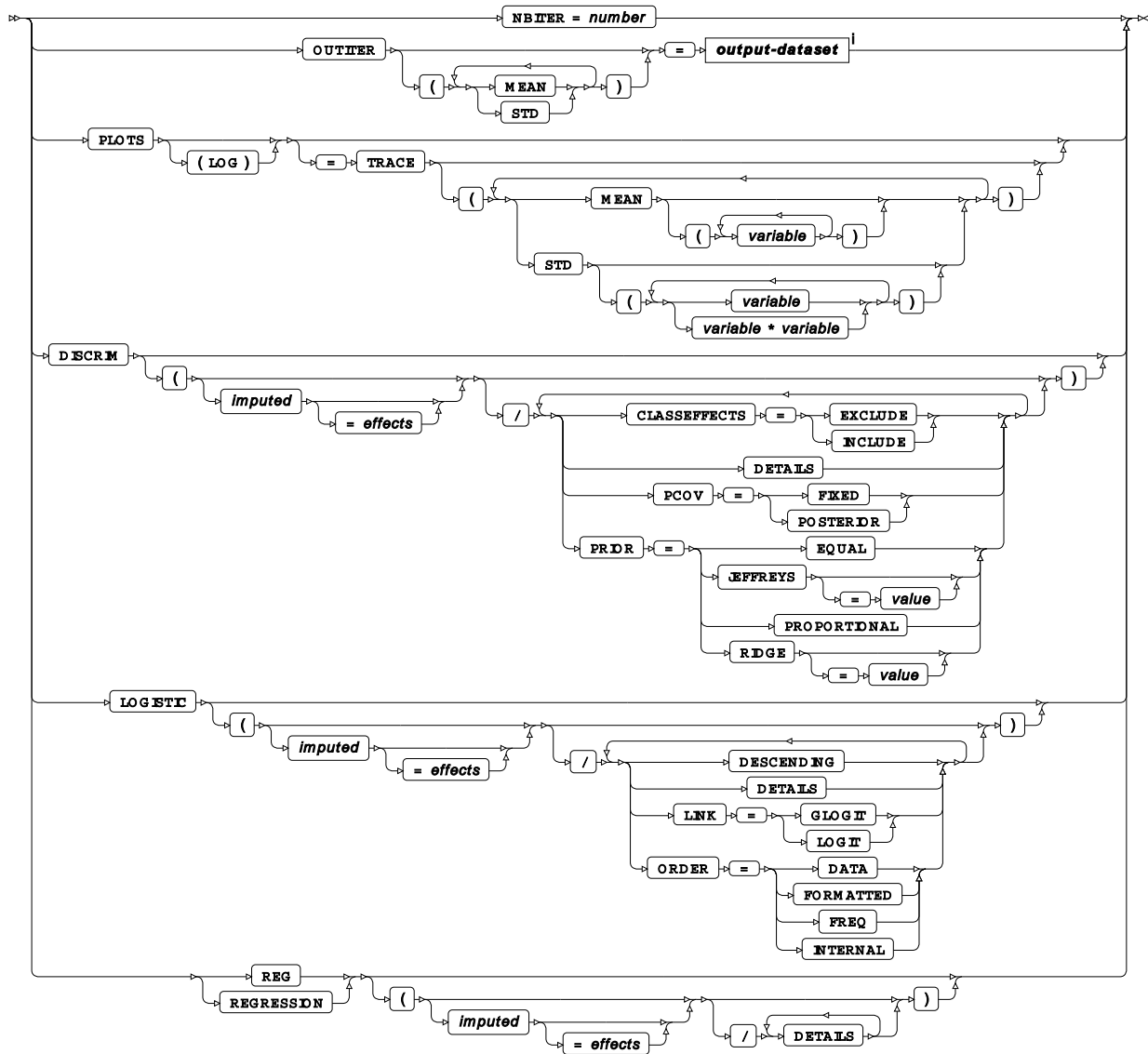
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

FCS



options



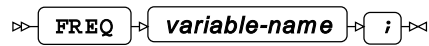
ⁱ See [Input dataset](#) (page 16).

FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ

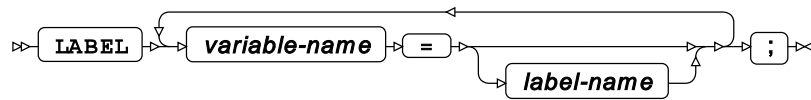


INFORMAT

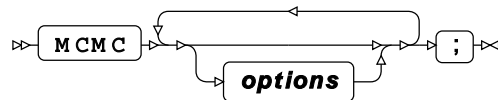


ⁱ See [Variable Lists](#) (page 32).

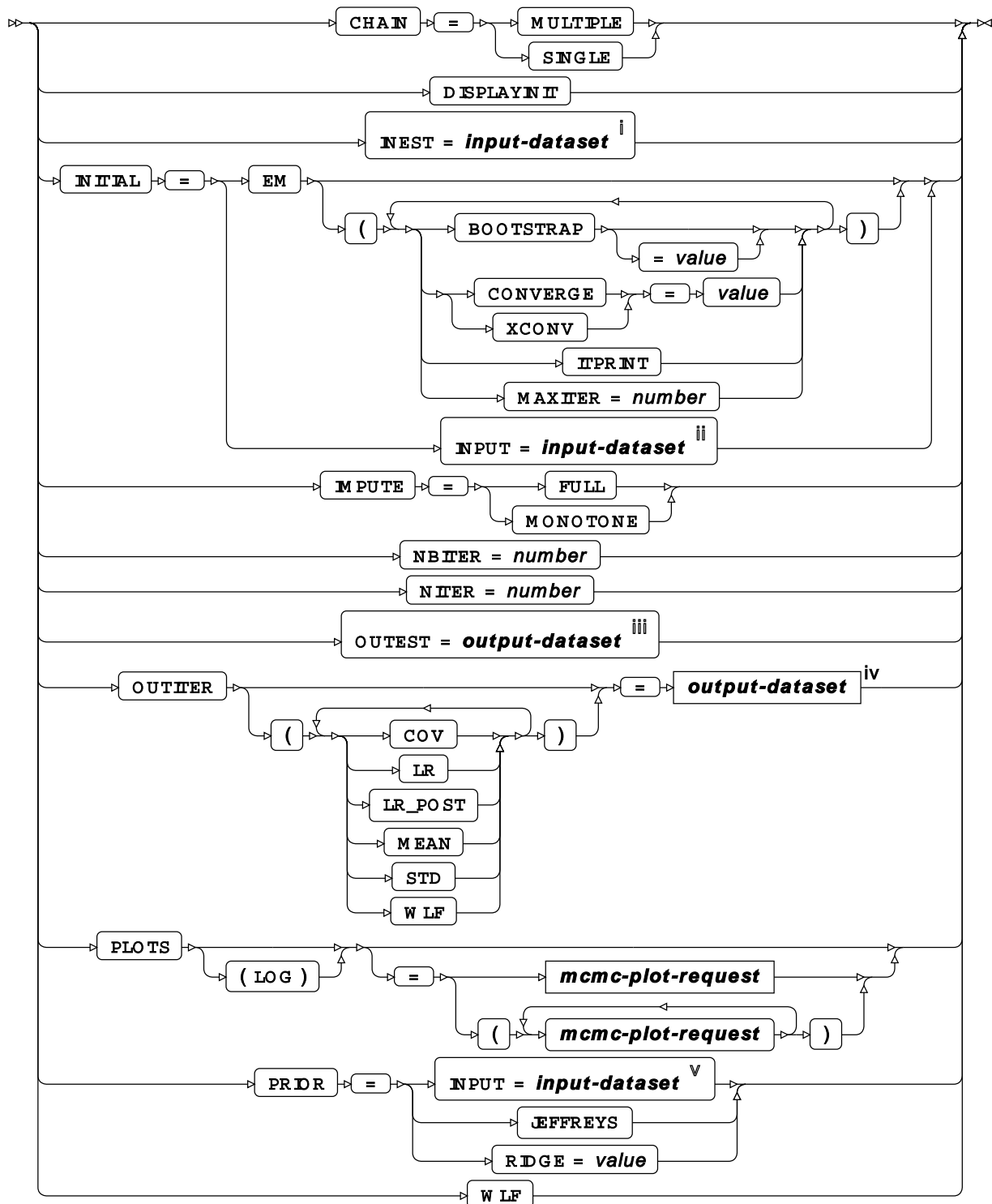
LABEL



MCMC



options



ⁱ See *Input dataset* [\[link\]](#) (page 16).

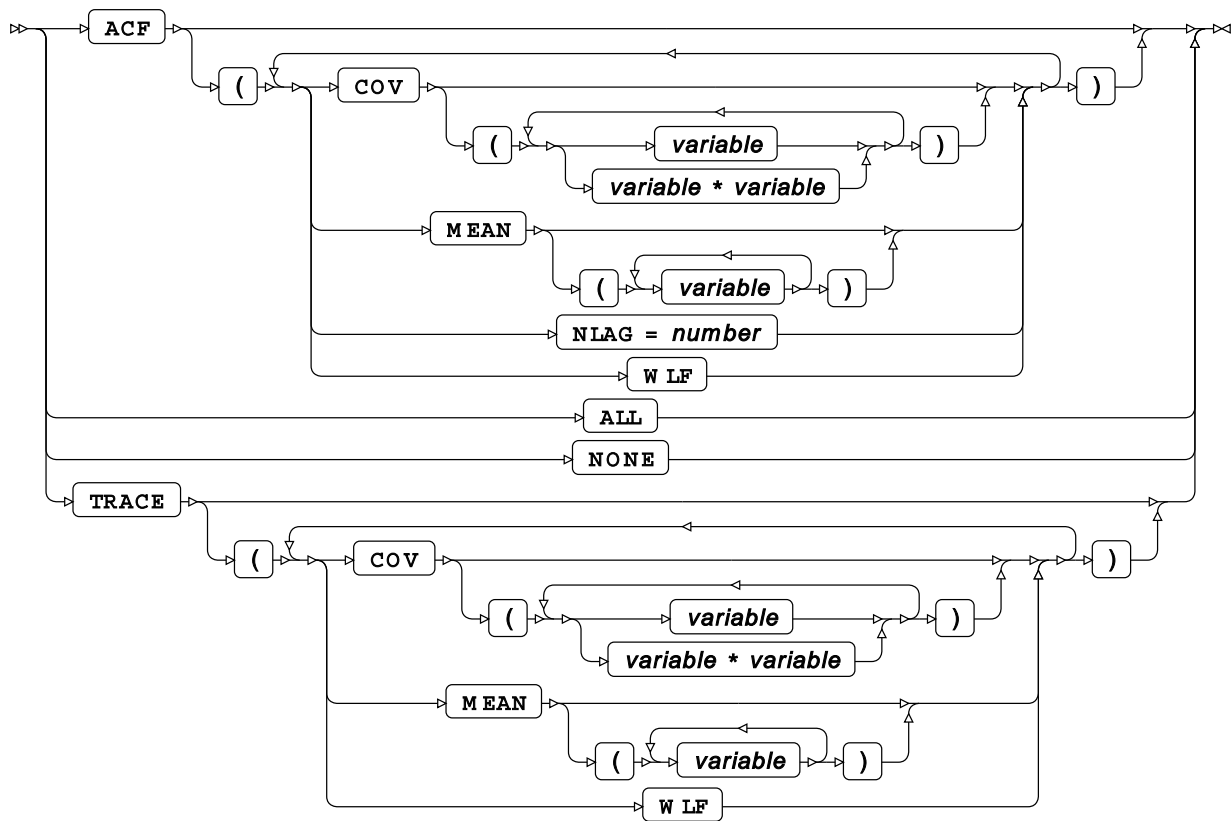
ⁱⁱ See *Input dataset* [\[link\]](#) (page 16).

iii See *Input dataset* [↗](#) (page 16).

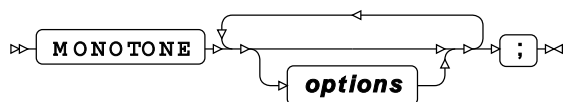
iv See *Input dataset* [↗](#) (page 16).

v See *Input dataset* [↗](#) (page 16).

mcmc-plot-request



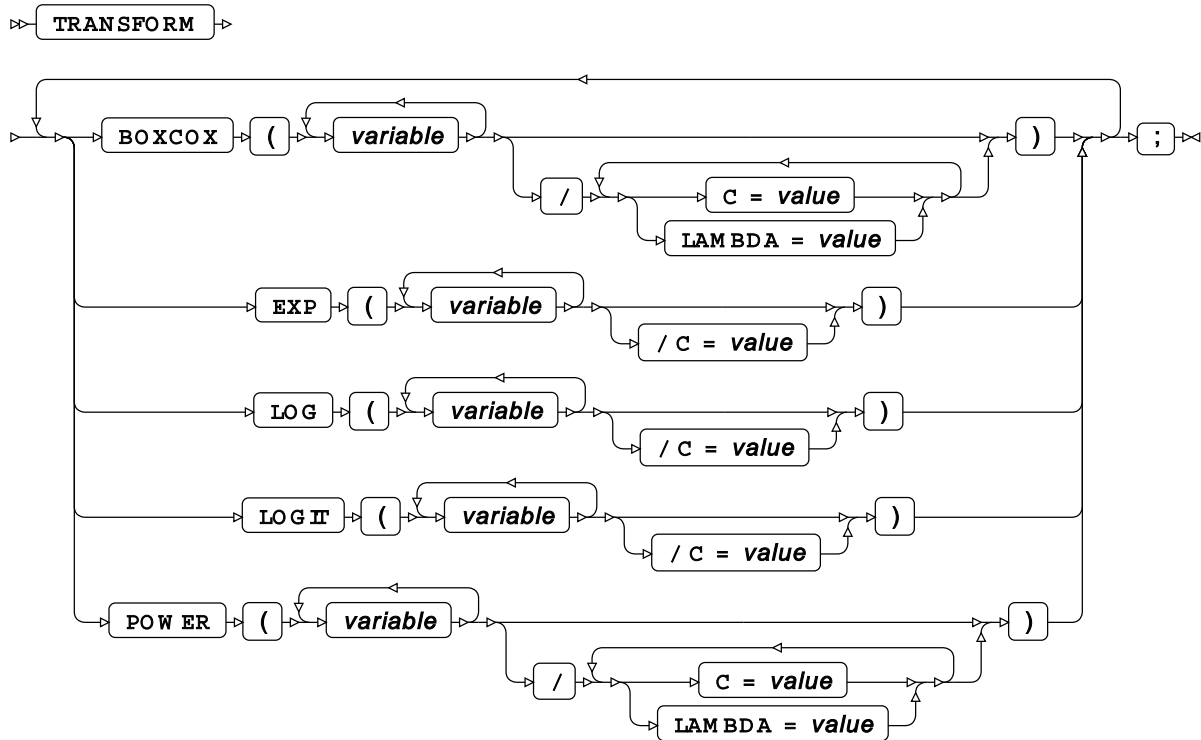
MONOTONE



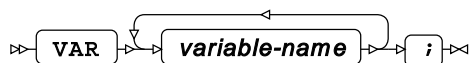
```

graph TD
    DESCENDING[DESCENDING] --> L1_1[ ]
    L1_1 --> L2_1["("]
    L2_1 --> L3_1["imputed"]
    L3_1 --> L4_1["= effects"]
    L4_1 --> L5_1["/"]
    L5_1 --> L6_1["CLASSEFFECTS"]
    L6_1 --> L7_1["="]
    L7_1 --> L8_1["EXCLUDE"]
    L8_1 --> L9_1["INCLUDE"]
    L9_1 --> L10_1["DETAILS"]
    L10_1 --> L11_1["PCOV"]
    L11_1 --> L12_1["="]
    L12_1 --> L13_1["FKED"]
    L13_1 --> L14_1["POSTERIOR"]
    L14_1 --> L15_1["PROR"]
    L15_1 --> L16_1["="]
    L16_1 --> L17_1["EQUAL"]
    L17_1 --> L18_1["JEFFREYS"]
    L18_1 --> L19_1["="]
    L19_1 --> L20_1["value"]
    L20_1 --> L21_1["PROPORTIONAL"]
    L21_1 --> L22_1["RIDGE"]
    L22_1 --> L23_1["="]
    L23_1 --> L24_1["value"]
    L24_1 --> L25_1[")"]
    L25_1 --> L26_1[ ]
    L26_1 --> LOGISTIC[LOGISTIC]
    LOGISTIC --> L27_1["("]
    L27_1 --> L28_1["imputed"]
    L28_1 --> L29_1["= effects"]
    L29_1 --> L30_1["/"]
    L30_1 --> L31_1["DESCENDING"]
    L31_1 --> L32_1["DETAILS"]
    L32_1 --> L33_1["LNK"]
    L33_1 --> L34_1["="]
    L34_1 --> L35_1["GLOGIT"]
    L35_1 --> L36_1["LOGIT"]
    L36_1 --> L37_1["ORDER"]
    L37_1 --> L38_1["="]
    L38_1 --> L39_1["DATA"]
    L39_1 --> L40_1["FORMATTED"]
    L40_1 --> L41_1["FREQ"]
    L41_1 --> L42_1["INTERNAL"]
    L42_1 --> L43_1[")"]
    L43_1 --> L44_1[ ]
    L44_1 --> REGRESSION[REGRESSION]
    REGRESSION --> L45_1["REG"]
    L45_1 --> L46_1["("]
    L46_1 --> L47_1["imputed"]
    L47_1 --> L48_1["= effects"]
    L48_1 --> L49_1["/"]
    L49_1 --> L50_1["DETAILS"]
    L50_1 --> L51_1[")"]
    L51_1 --> L52_1[ ]
  
```

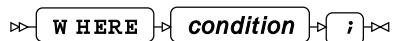
TRANSFORM



VAR



WHERE



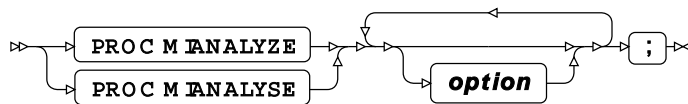
MIANALYZE procedure

Supported statements

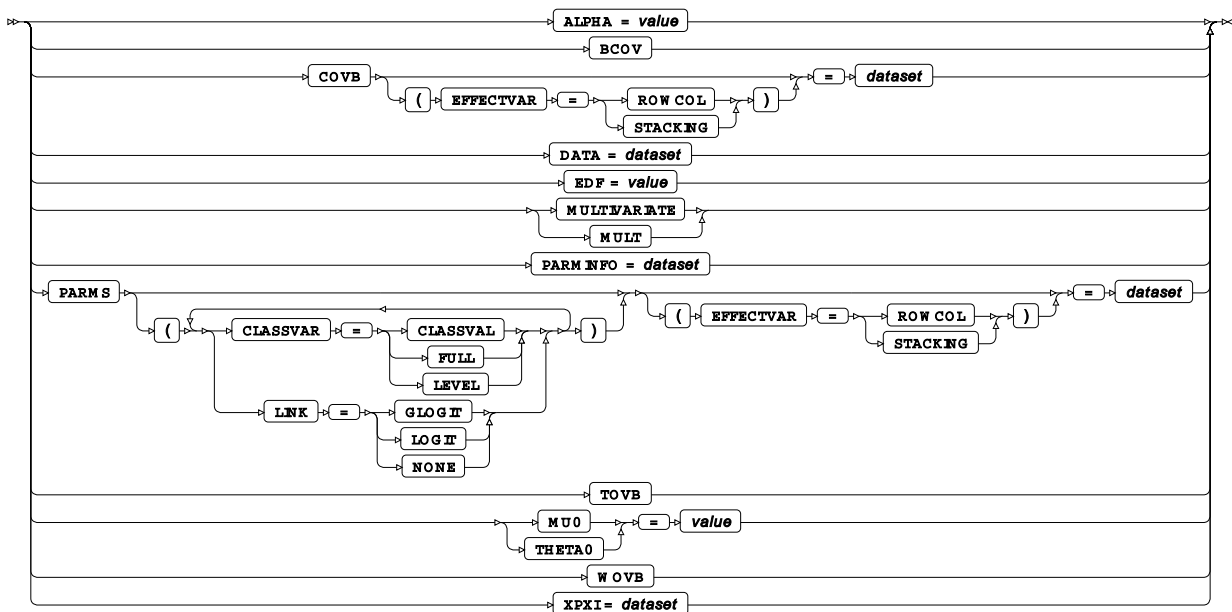
- *PROC MIANALYZE* [↗](#) (page 2962)
- *ATTRIB* [↗](#) (page 2963)

- [BY](#) (page 2963)
- [CLASS](#) (page 2963)
- [FORMAT](#) (page 2963)
- [INFORMAT](#) (page 2963)
- [LABEL](#) (page 2964)
- [MODELEFFECTS](#) (page 2964)
- [STDERR](#) (page 2964)
- [TEST](#) (page 2964)
- [WHERE](#) (page 2965)

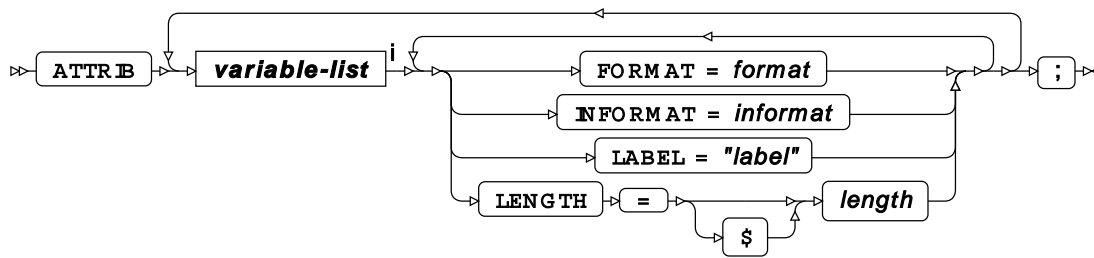
PROC MIANALYZE



option

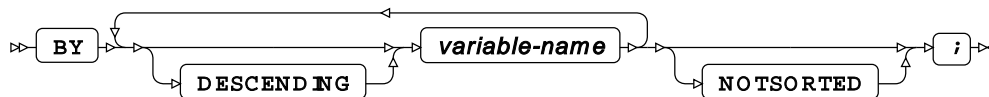


ATTRIB

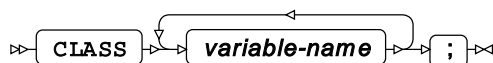


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



CLASS



FORMAT



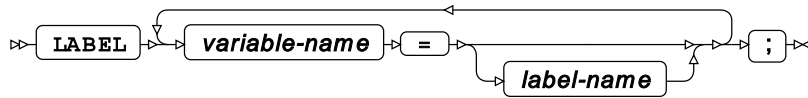
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

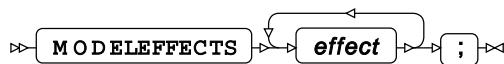


ⁱ See *Variable Lists* [↗](#) (page 32).

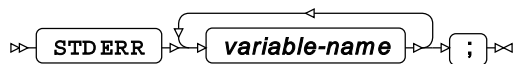
LABEL



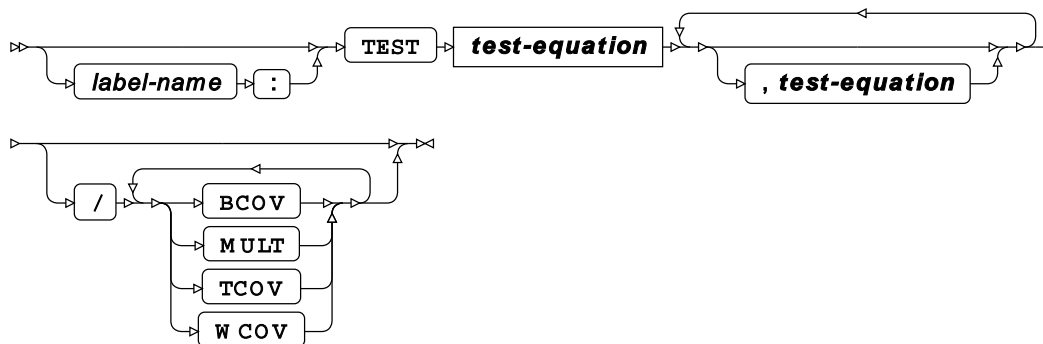
MODELEFFECTS



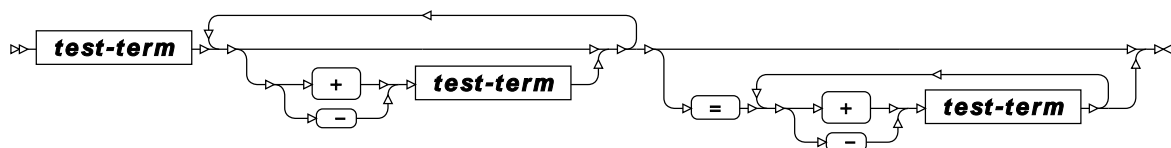
STDERR



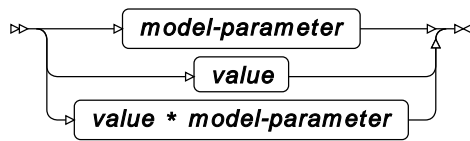
TEST



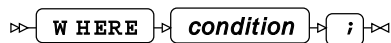
test-equation



test-term



WHERE

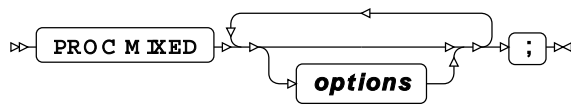


MIXED procedure

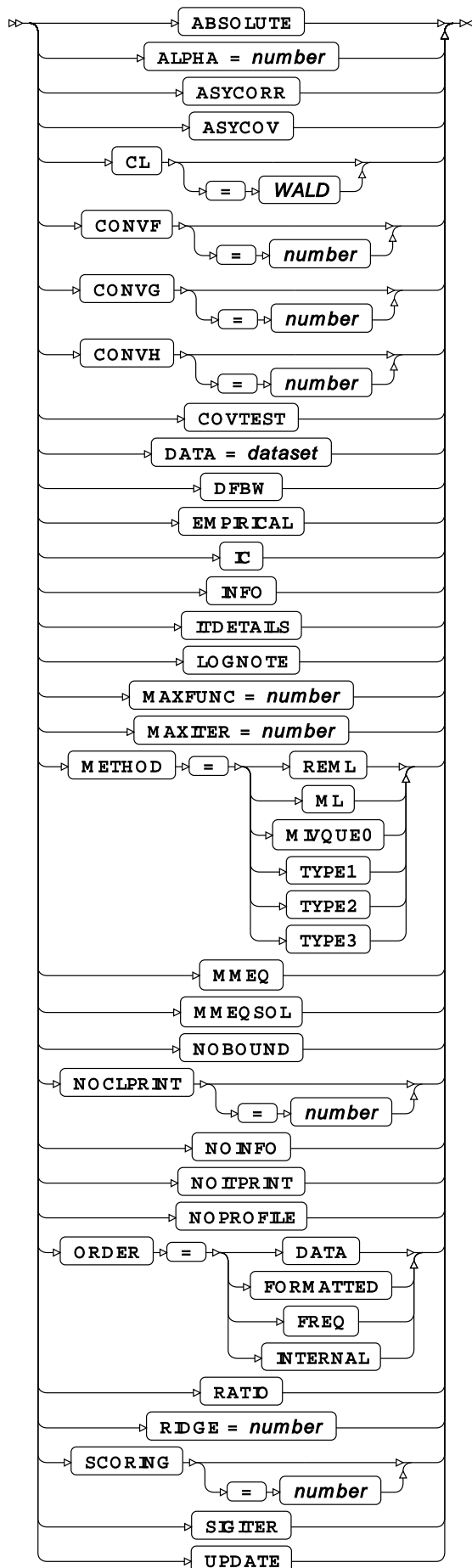
Supported statements

- *PROC MIXED* [↗](#) (page 2966)
- *ATTRIB* [↗](#) (page 2968)
- *BY* [↗](#) (page 2968)
- *CLASS* [↗](#) (page 2968)
- *CODE* [↗](#) (page 2968)
- *CONTRAST* [↗](#) (page 2969)
- *ESTIMATE* [↗](#) (page 2970)
- *FORMAT* [↗](#) (page 2971)
- *ID* [↗](#) (page 2971)
- *INFORMAT* [↗](#) (page 2971)
- *LABEL* [↗](#) (page 2972)
- *LSMEANS* [↗](#) (page 2972)
- *MODEL* [↗](#) (page 2974)
- *RANDOM* [↗](#) (page 2976)
- *REPEATED* [↗](#) (page 2978)
- *WEIGHT* [↗](#) (page 2979)
- *WHERE* [↗](#) (page 2979)

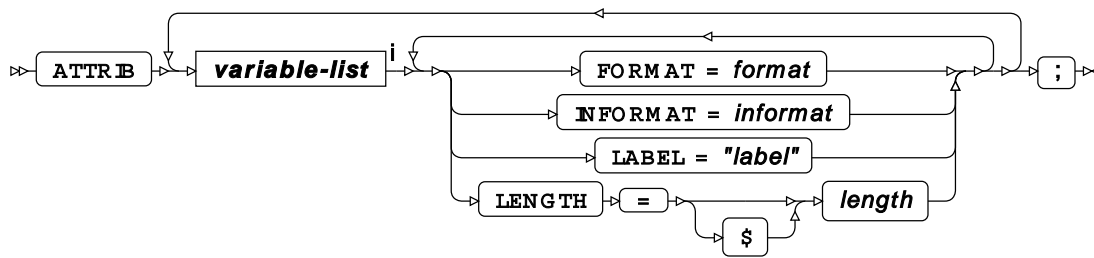
PROC MIXED



options

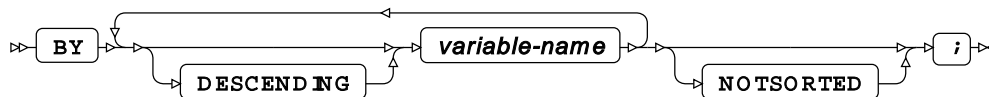


ATTRIB

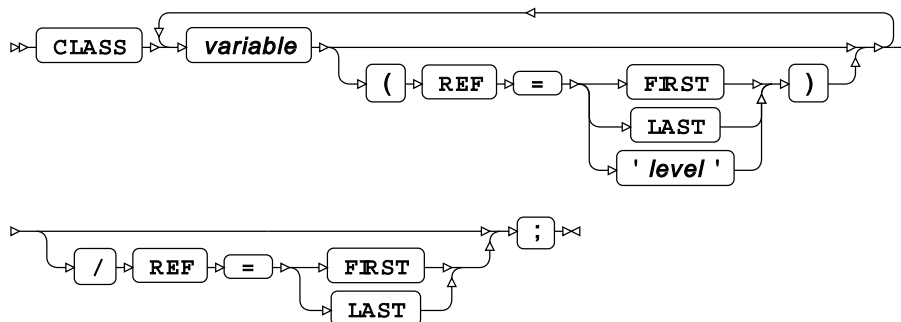


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



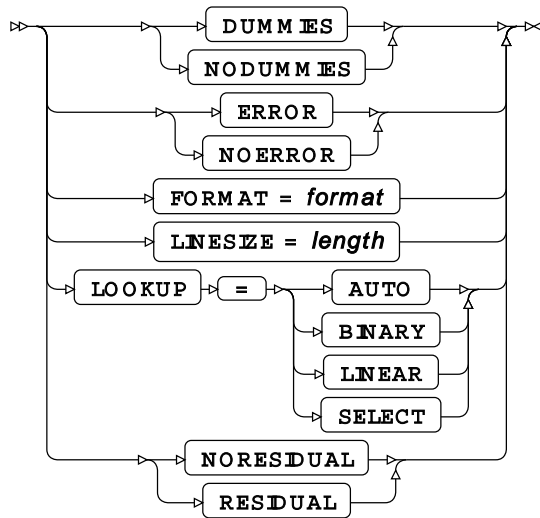
CLASS



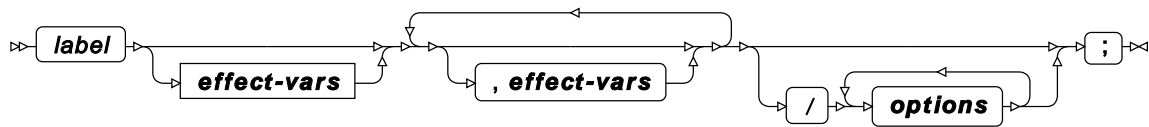
CODE



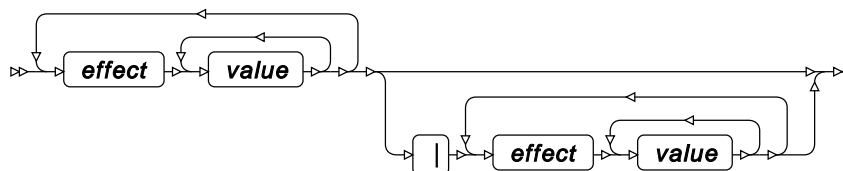
options



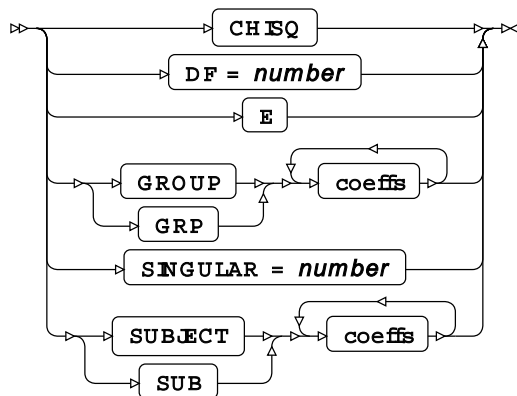
CONTRAST



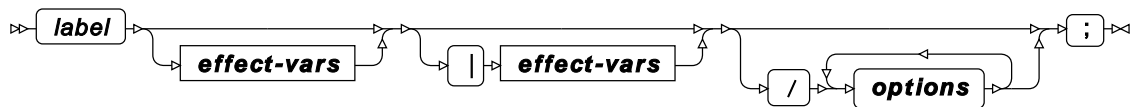
effect-vars



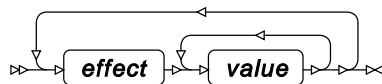
options



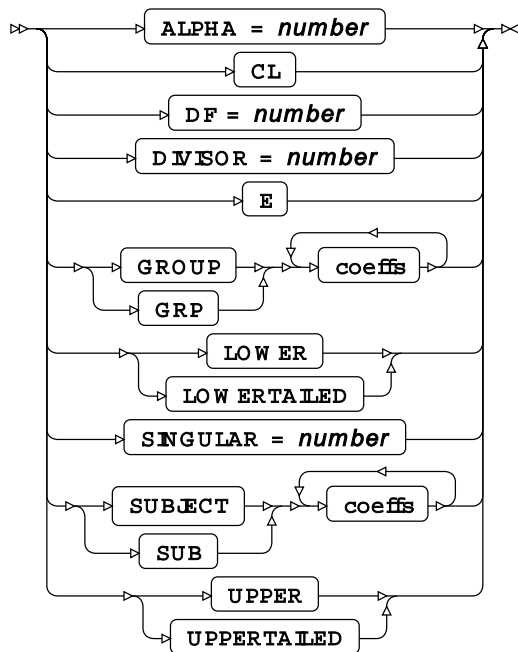
ESTIMATE



effect-vars



options

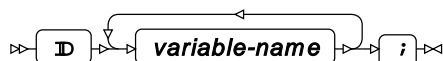


FORMAT



ⁱ See [Variable Lists](#) (page 32).

ID

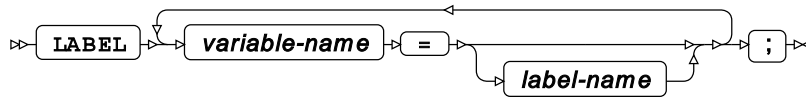


INFORMAT

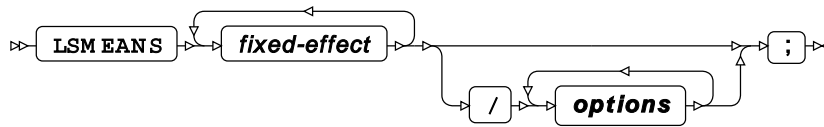


ⁱ See *Variable Lists* [↗](#) (page 32).

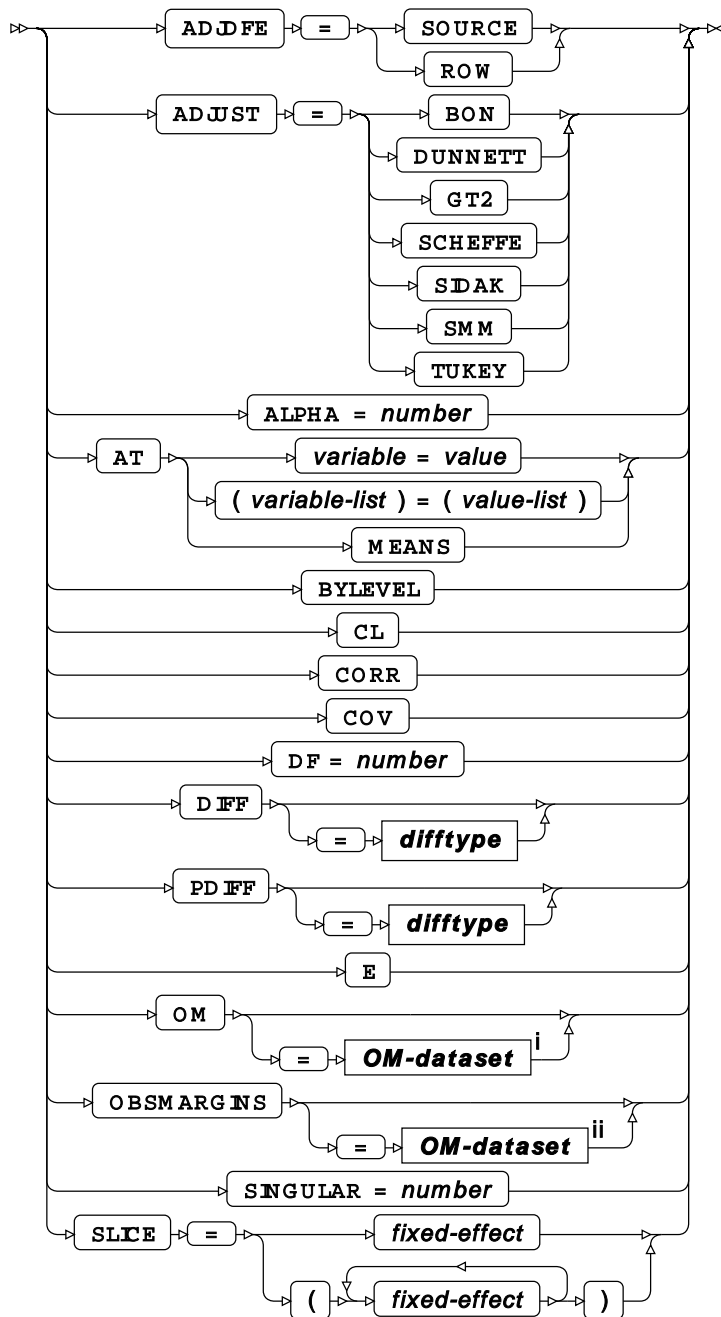
LABEL



LSMEANS



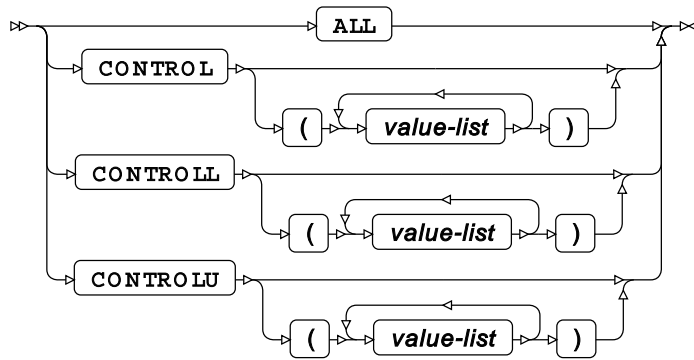
options



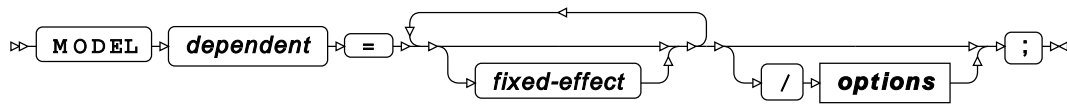
ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

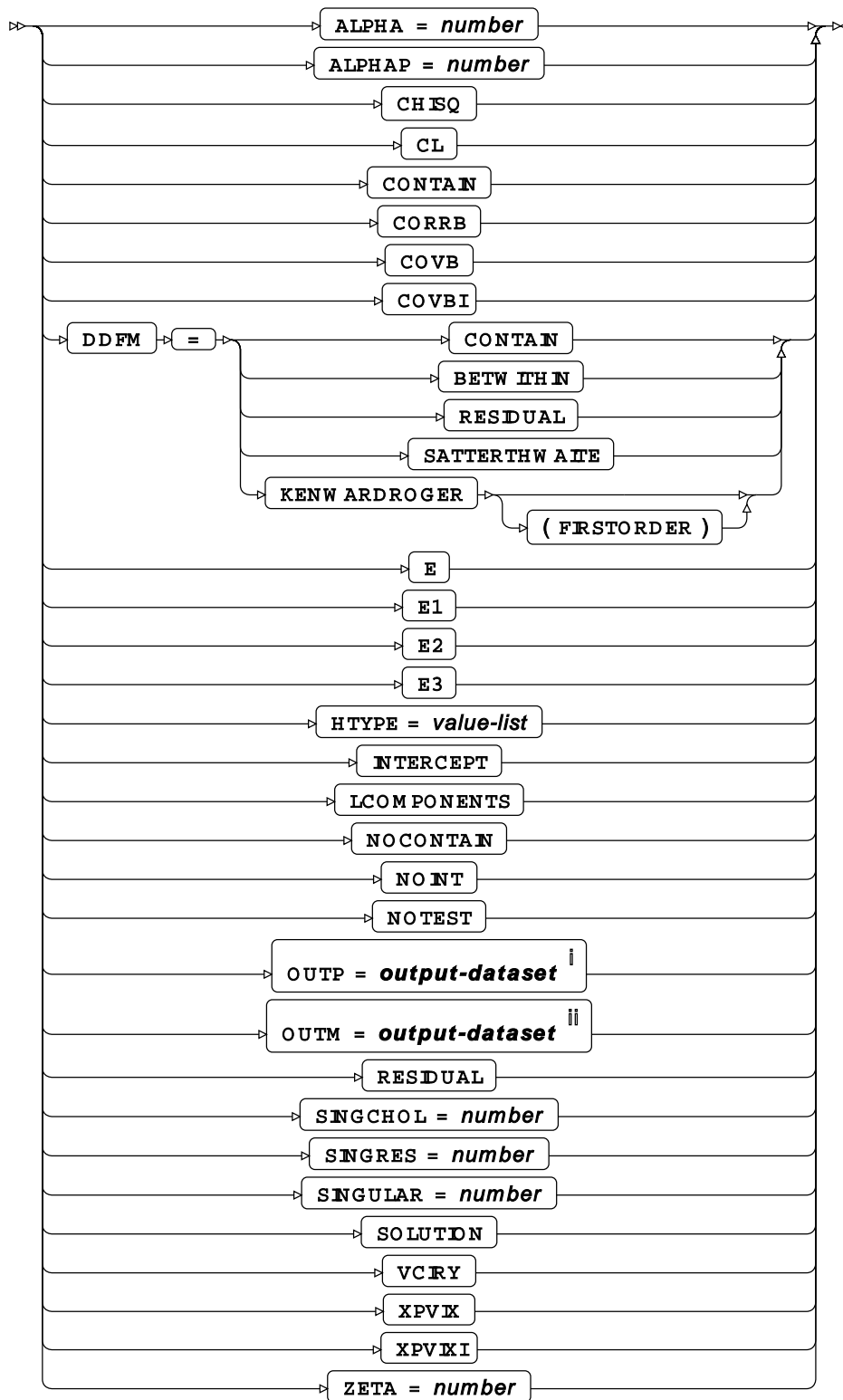
difftype



MODEL

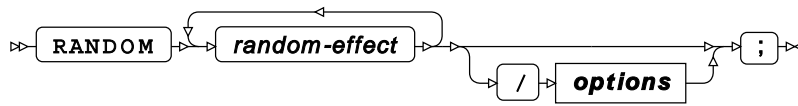


options

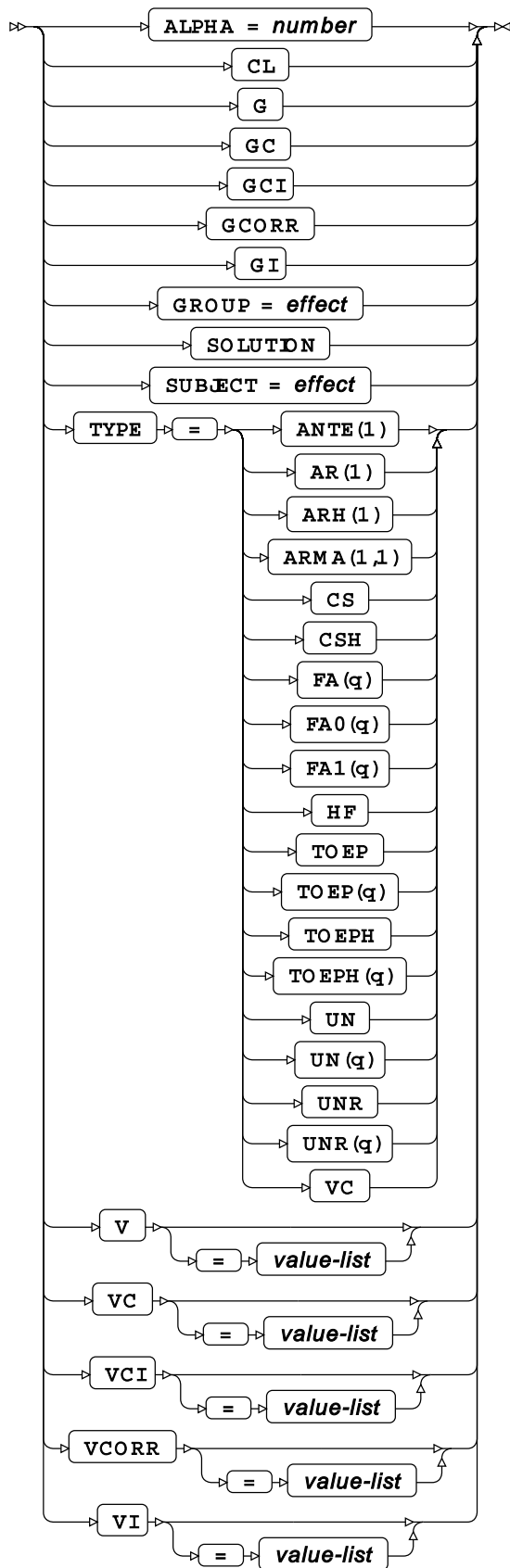
ⁱ See [Output dataset](#) (page 16).

- ii See *Output dataset* [↗](#) (page 16).

RANDOM



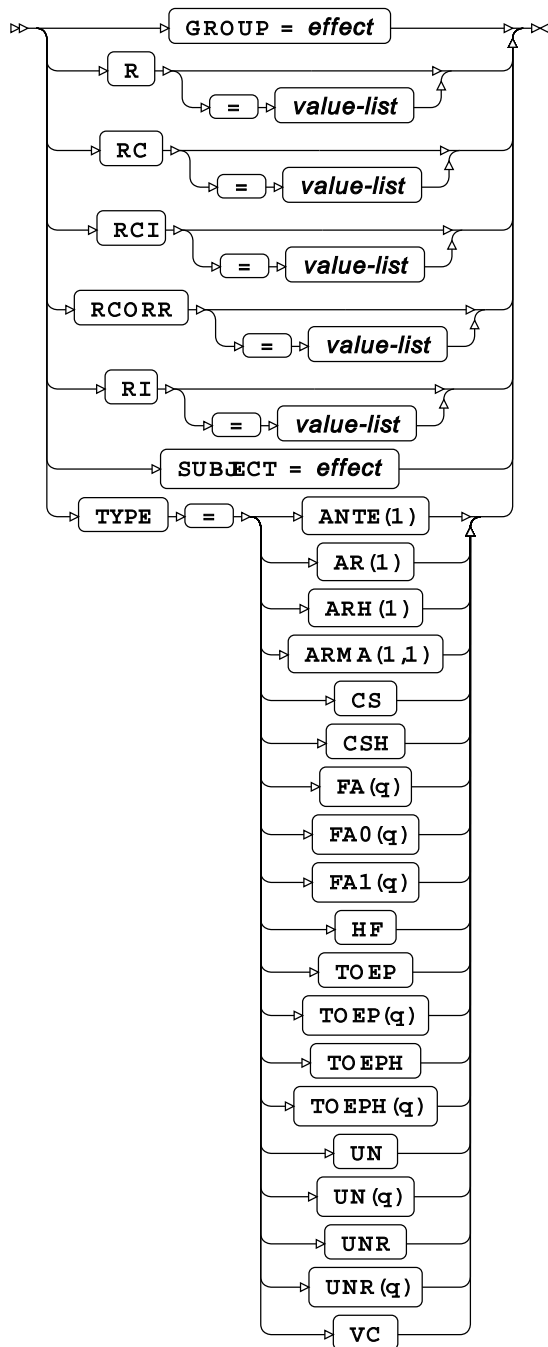
options



REPEATED



options



WEIGHT



WHERE

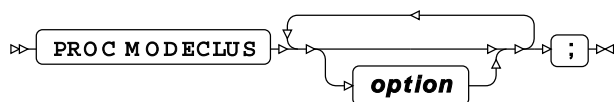


MODECLUS procedure

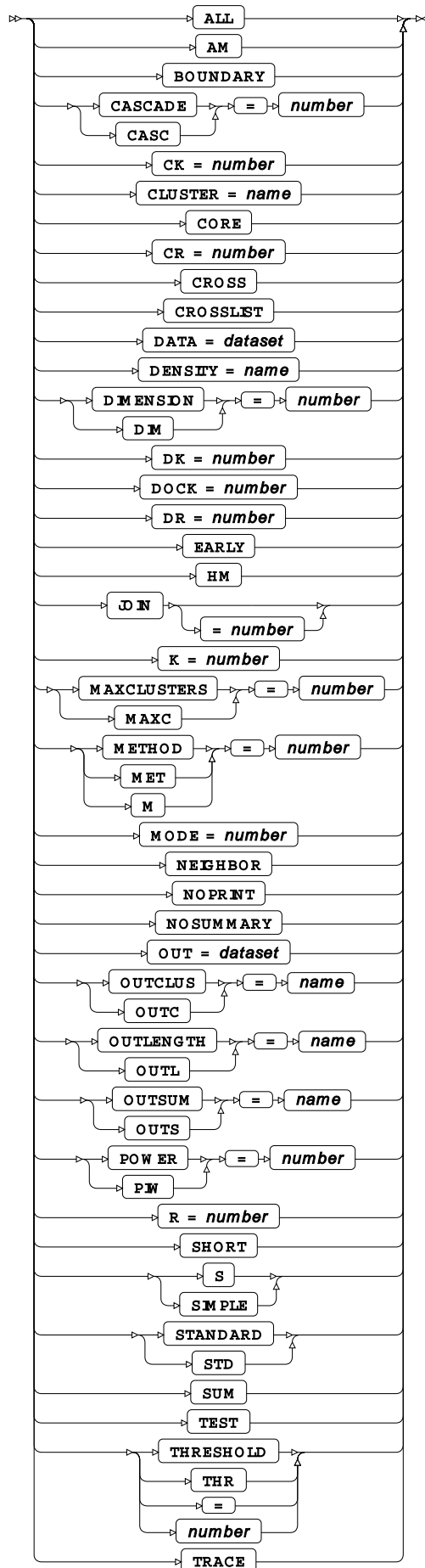
Supported statements

- *PROC MODECLUS* [↗](#) (page 2979)
- *ATTRIB* [↗](#) (page 2981)
- *BY* [↗](#) (page 2981)
- *FORMAT* [↗](#) (page 2981)
- *FREQ* [↗](#) (page 2981)
- *ID* [↗](#) (page 2981)
- *INFORMAT* [↗](#) (page 2982)
- *FORMAT* [↗](#) (page 2981)
- *VAR* [↗](#) (page 2982)
- *WHERE* [↗](#) (page 2982)

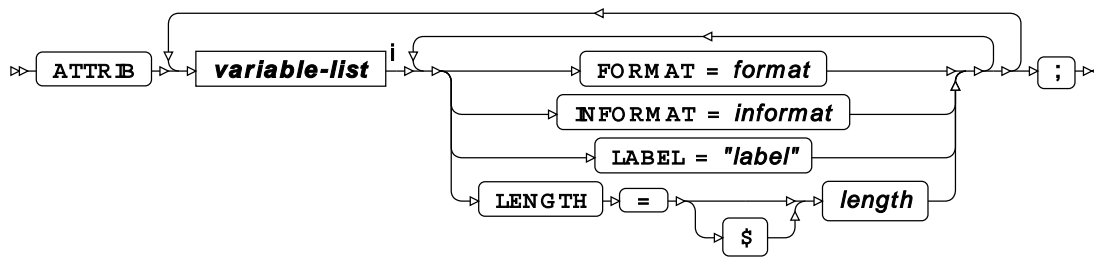
PROC MODECLUS



option

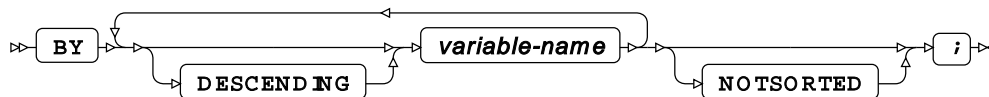


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

BY



FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ



ID

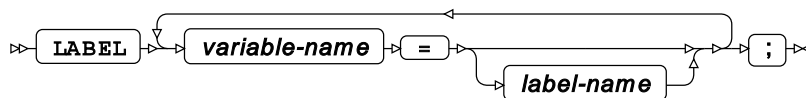


INFORMAT

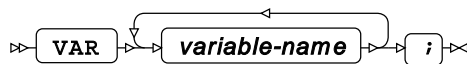


ⁱ See *Variable Lists* [↗](#) (page 32).

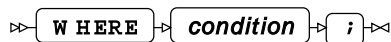
LABEL



VAR



WHERE



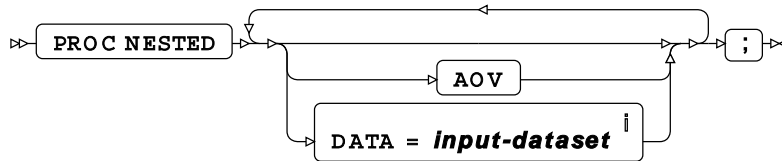
NESTED procedure

Supported statements

- *PROC NESTED* [↗](#) (page 2983)
- *ATTRIB* [↗](#) (page 2983)
- *BY* [↗](#) (page 2983)
- *CLASS* [↗](#) (page 2983)
- *FORMAT* [↗](#) (page 2984)
- *INFORMAT* [↗](#) (page 2984)
- *LABEL* [↗](#) (page 2984)
- *VAR* [↗](#) (page 2984)

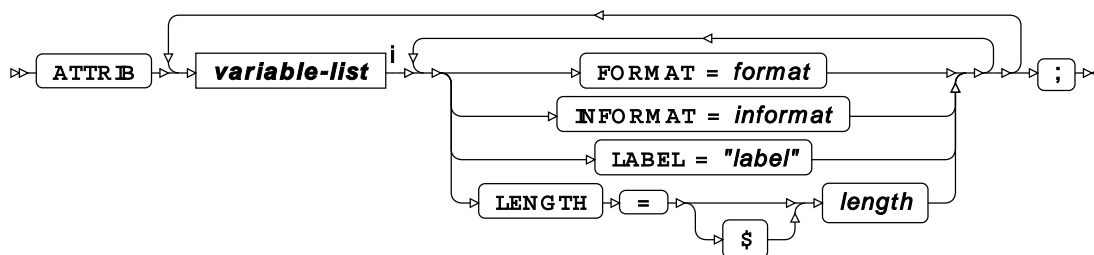
- [WHERE](#) (page 2984)

PROC NESTED



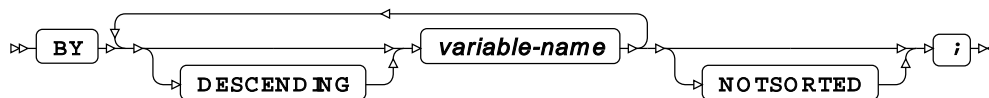
ⁱ See [Input dataset](#) (page 16).

ATTRIB

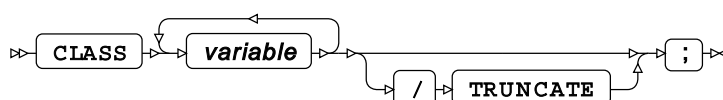


ⁱ See [Variable Lists](#) (page 32).

BY



CLASS



FORMAT



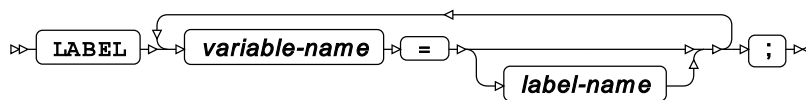
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

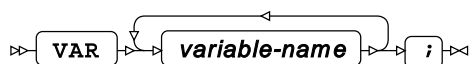


ⁱ See *Variable Lists* [↗](#) (page 32).

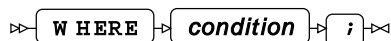
LABEL



VAR



WHERE

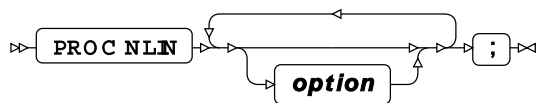


NLIN procedure

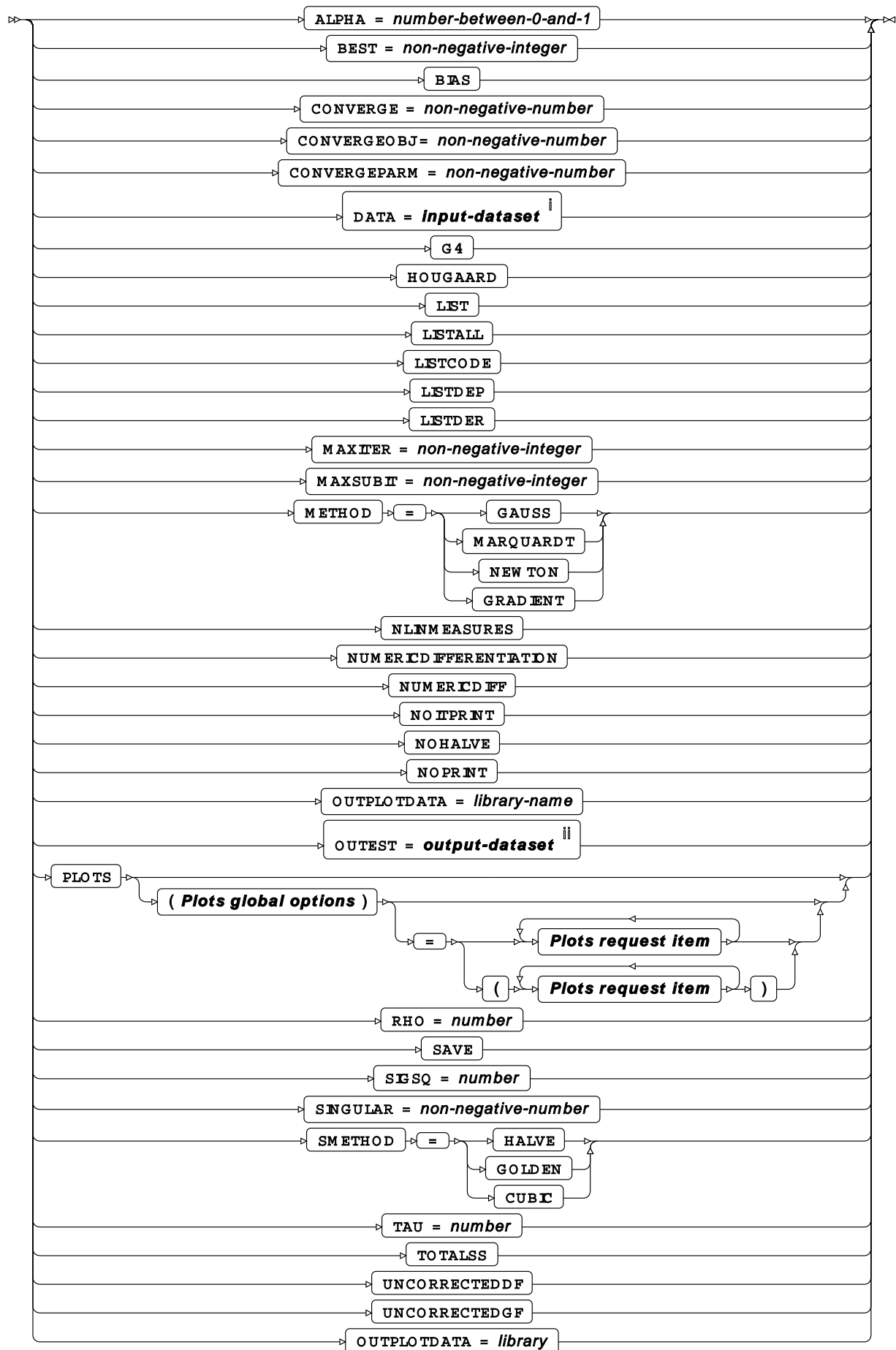
Supported statements

- *PROC NLIN* [↗](#) (page 2985)
- *ATTRIB* [↗](#) (page 2989)
- *BOUNDS* [↗](#) (page 2990)
- *BY* [↗](#) (page 2990)
- *CONTROL* [↗](#) (page 2990)
- *DER* [↗](#) (page 2991)
- *FORMAT* [↗](#) (page 2991)
- *ID* [↗](#) (page 2991)
- *INFORMAT* [↗](#) (page 2991)
- *LABEL* [↗](#) (page 2991)
- *MODEL* [↗](#) (page 2992)
- *OUTPUT* [↗](#) (page 2992)
- *PARAMETERS* [↗](#) (page 2994)
- *RETAIN* [↗](#) (page 2994)
- *VAR* [↗](#) (page 2994)
- *WEIGHT* [↗](#) (page 2994)
- *WHERE* [↗](#) (page 2994)

PROC NLIN

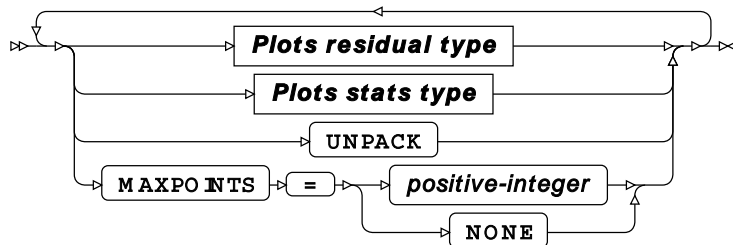


option

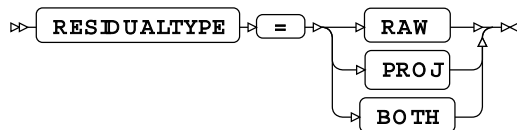


- i See *Input dataset* [↗](#) (page 16).
- ii See *Output dataset* [↗](#) (page 16).

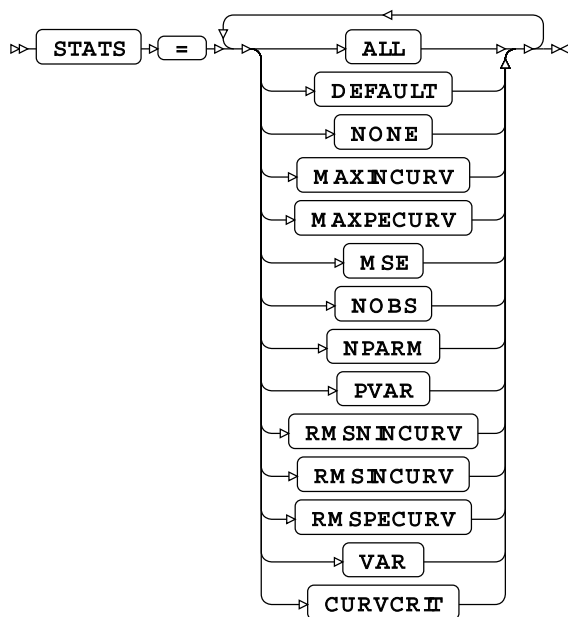
Plots global options



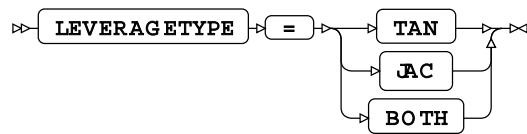
Plots residual type



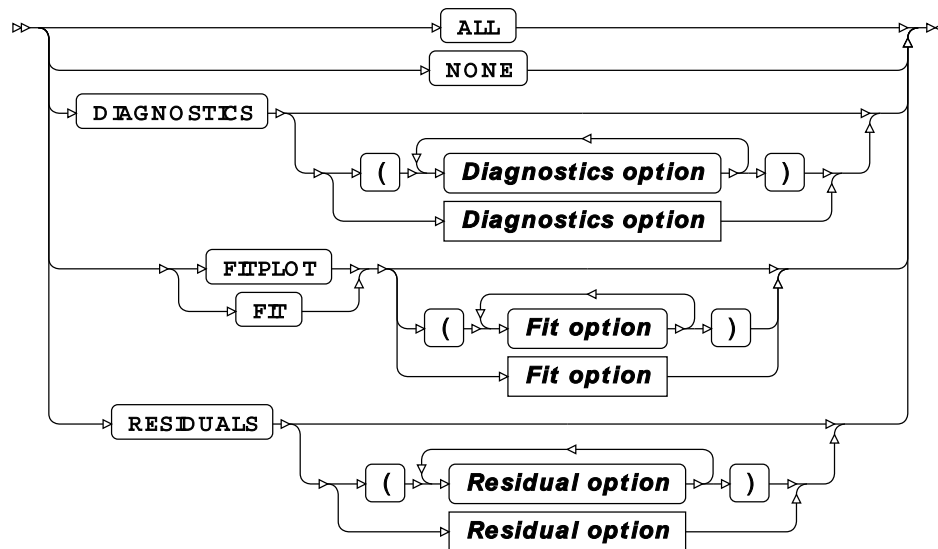
Plots stats type



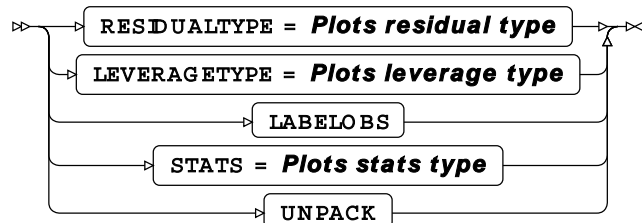
Plots leverage type



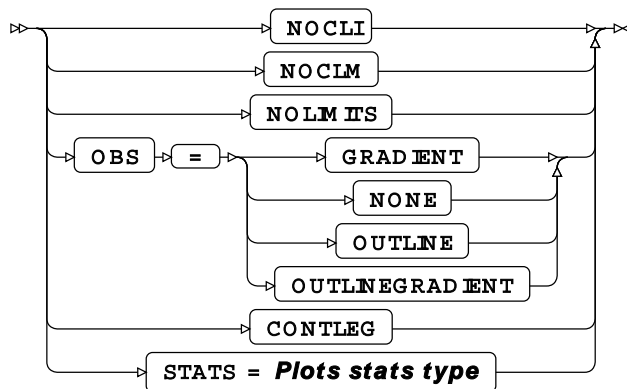
Plots request item



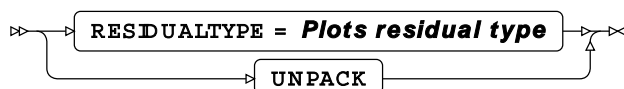
Diagnostics option



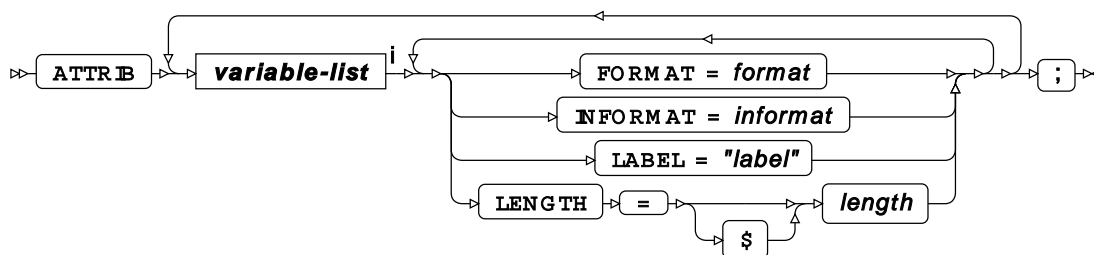
Fit option



Residual option

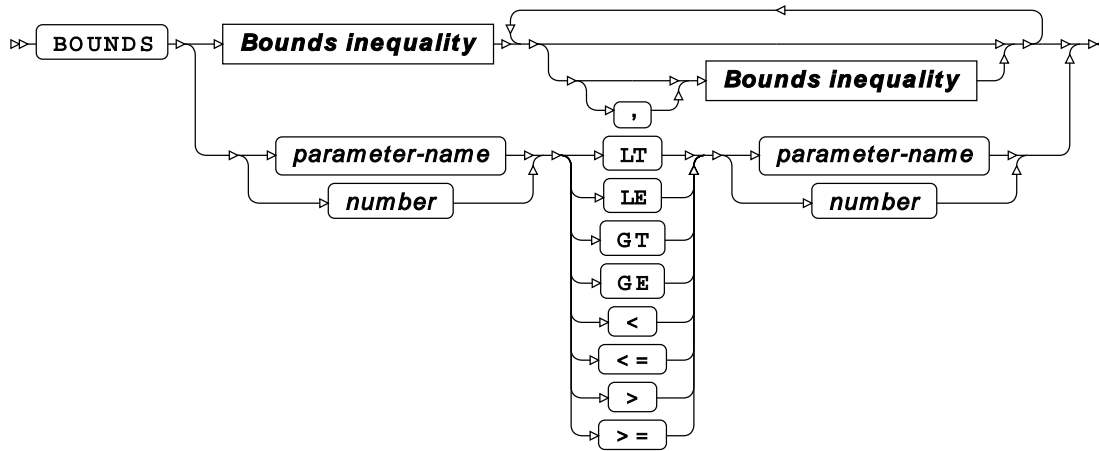


ATTRIB

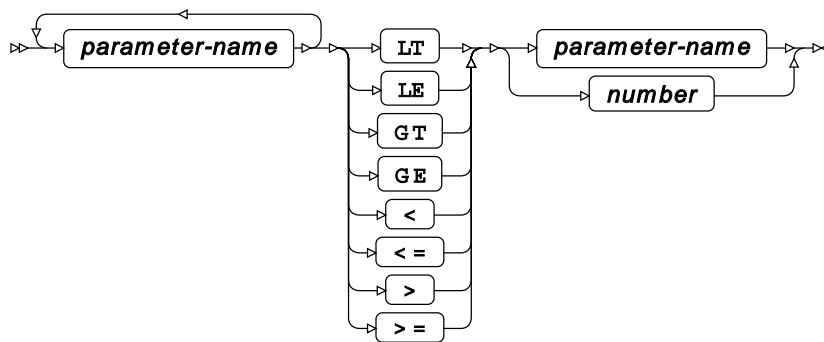


ⁱ See [Variable Lists](#) (page 32).

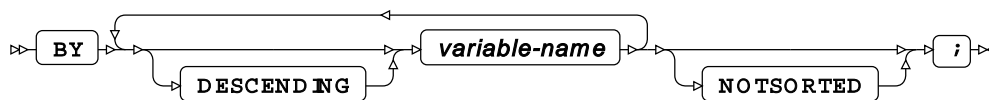
BOUNDS



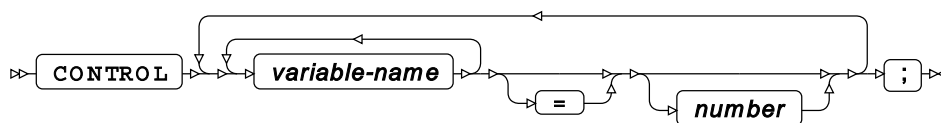
Bounds inequality



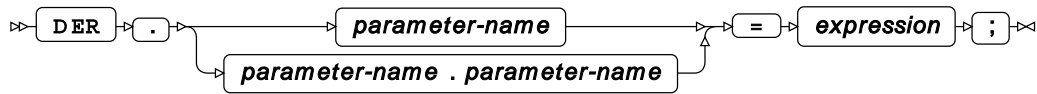
BY



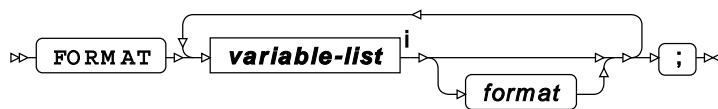
CONTROL



DER

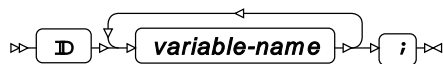


FORMAT



ⁱ See [Variable Lists](#) (page 32).

ID

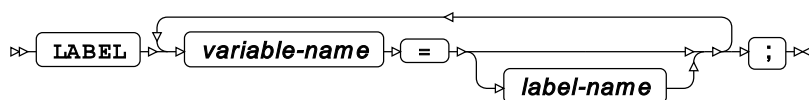


INFORMAT

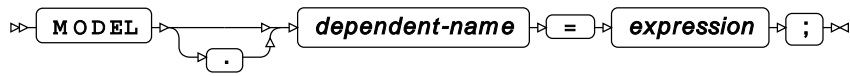


ⁱ See [Variable Lists](#) (page 32).

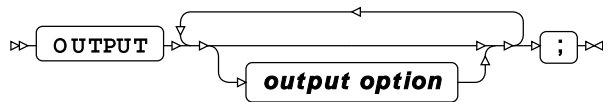
LABEL



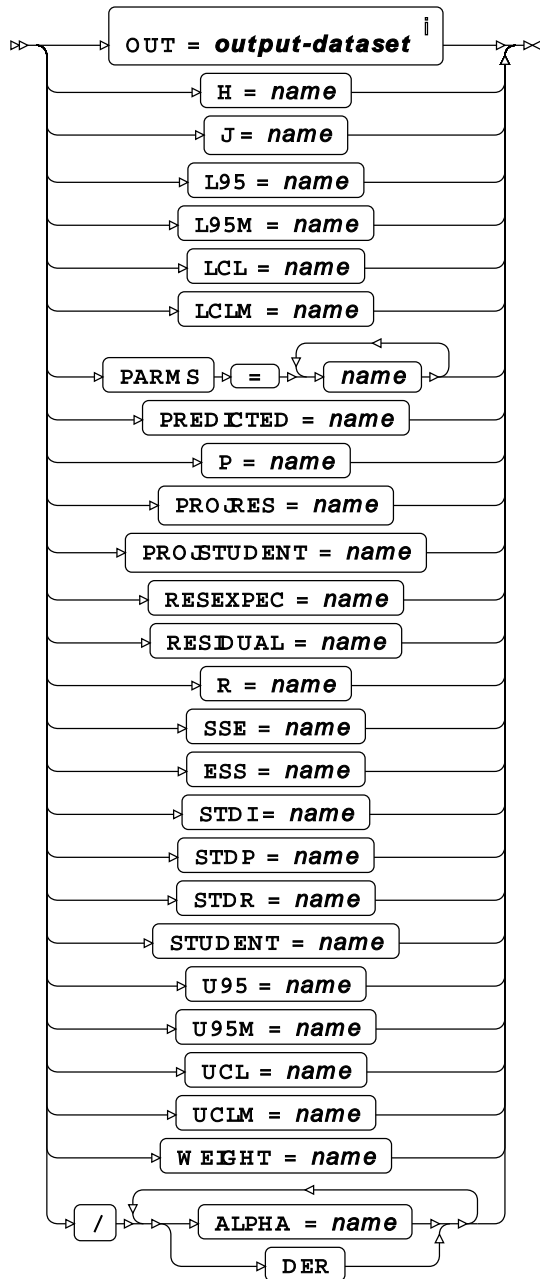
MODEL



OUTPUT

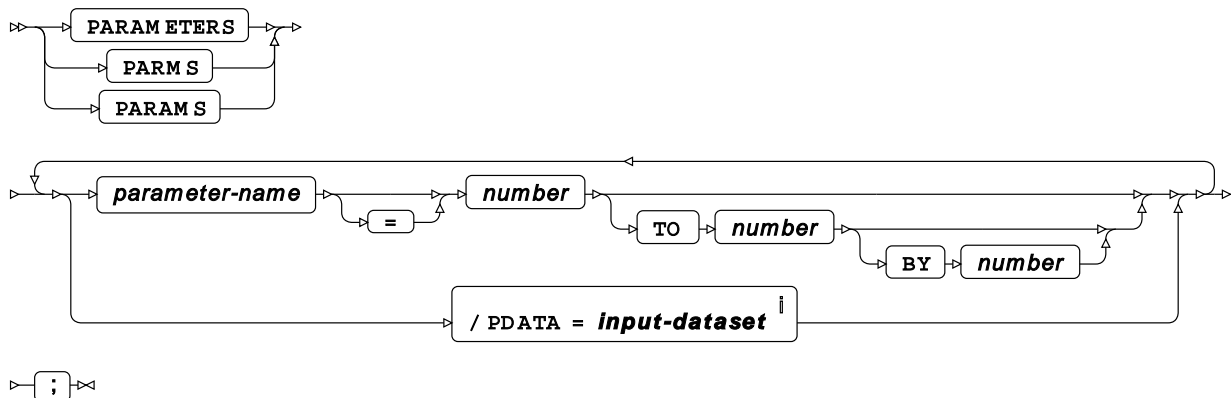


output option



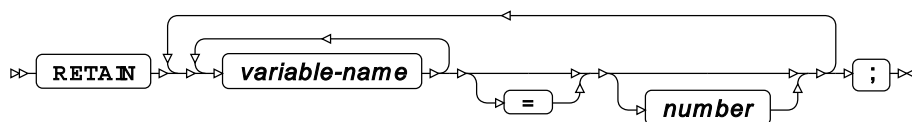
ⁱ See *Output dataset* [↗](#) (page 16).

PARAMETERS

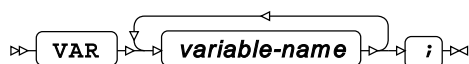


ⁱ See *Input dataset* [↗](#) (page 16).

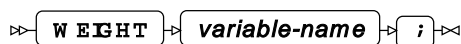
RETAIN



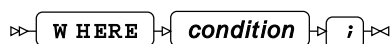
VAR



WEIGHT



WHERE

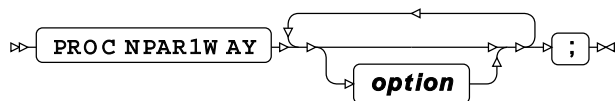


NPAR1WAY procedure

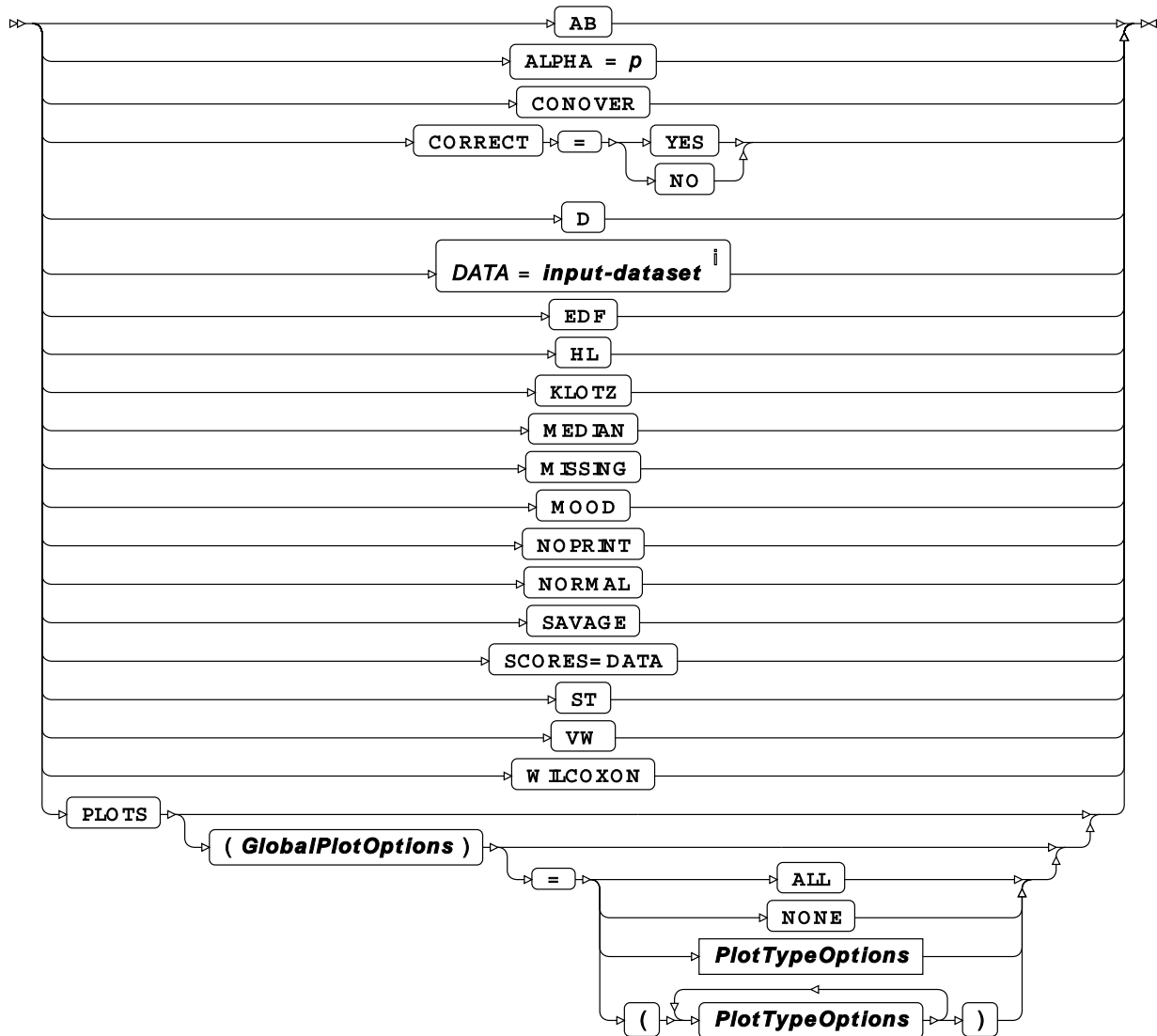
Supported statements

- *PROC NPAR1WAY* [↗](#) (page 2995)
- *ATTRIB* [↗](#) (page 2997)
- *BY* [↗](#) (page 2998)
- *CLASS* [↗](#) (page 2998)
- *EXACT* [↗](#) (page 2998)
- *FORMAT* [↗](#) (page 2999)
- *ID* [↗](#) (page 2999)
- *INFORMAT* [↗](#) (page 2999)
- *LABEL* [↗](#) (page 2999)
- *FREQ* [↗](#) (page 2999)
- *OUTPUT* [↗](#) (page 3000)
- *VAR* [↗](#) (page 3000)
- *WHERE* [↗](#) (page 3000)

PROC NPAR1WAY

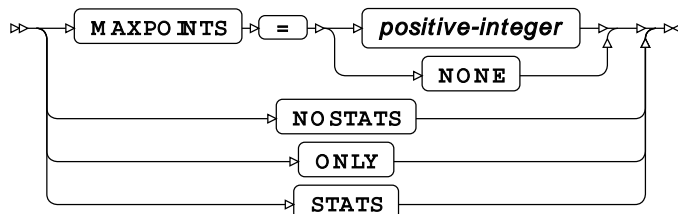


option

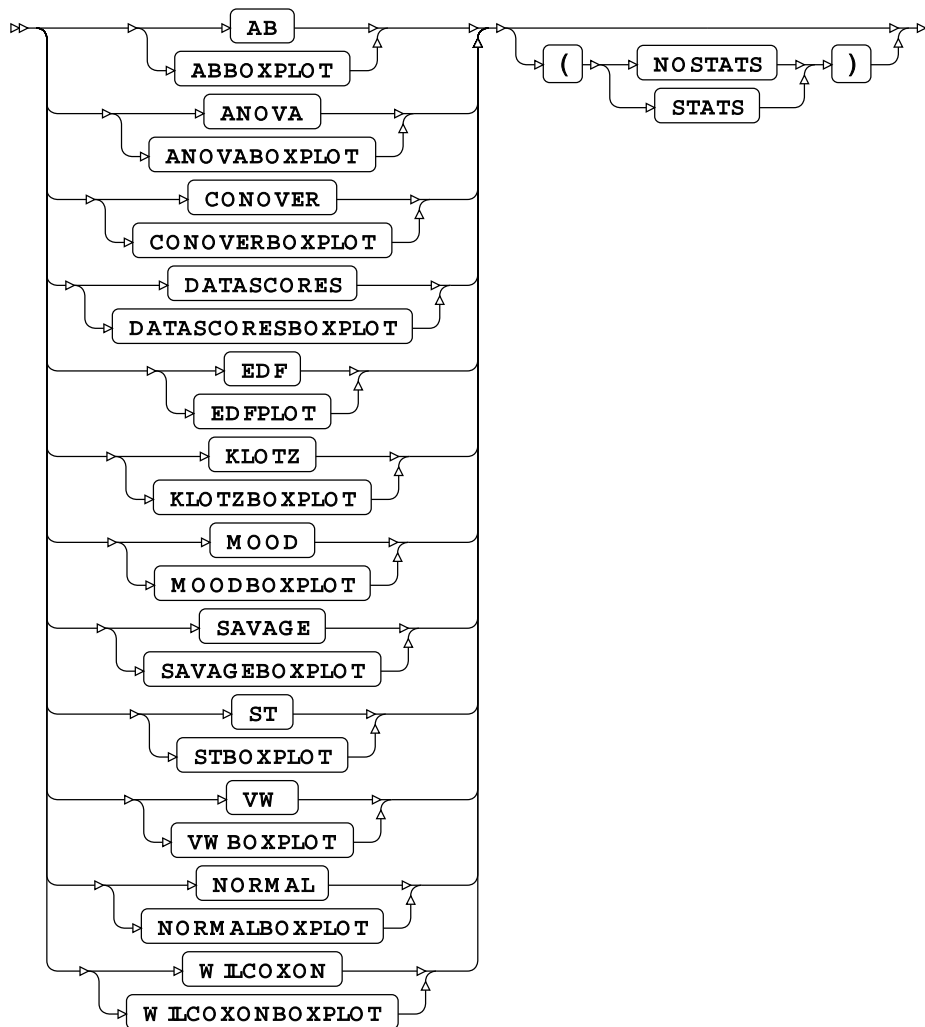


ⁱ See *Input dataset* [↗](#) (page 16).

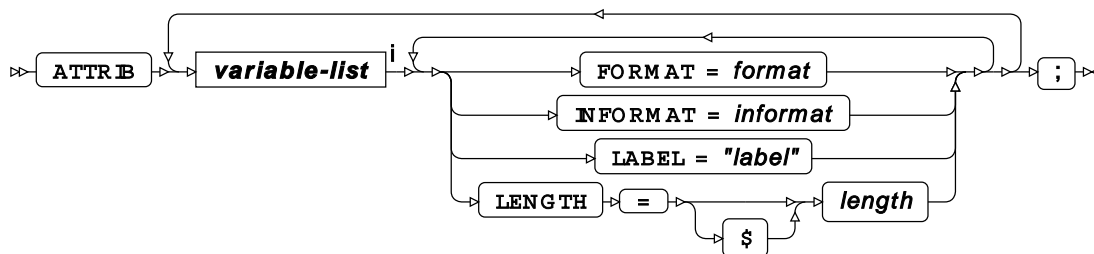
GlobalPlotOptions



PlotTypeOptions

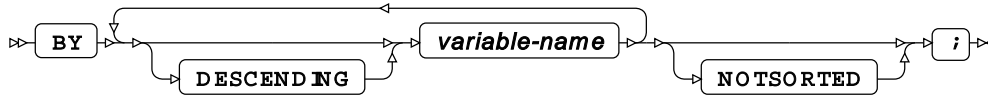


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

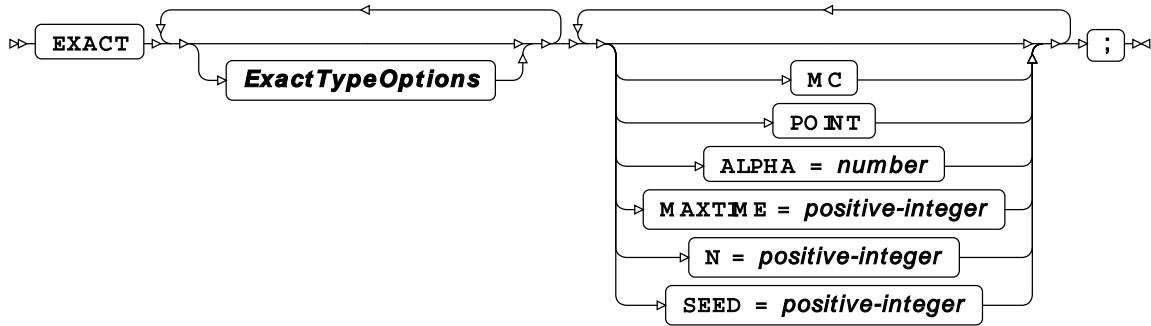
BY



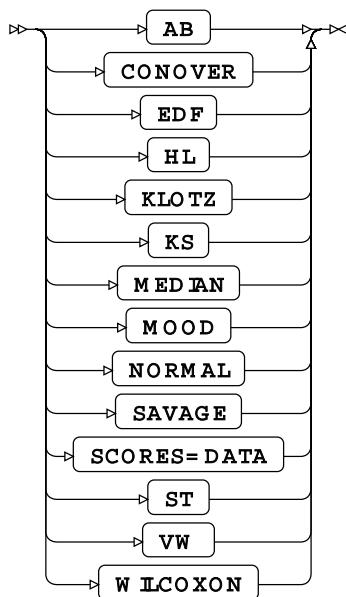
CLASS



EXACT



ExactTypeOptions

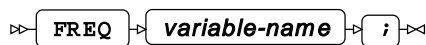


FORMAT

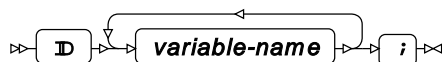


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

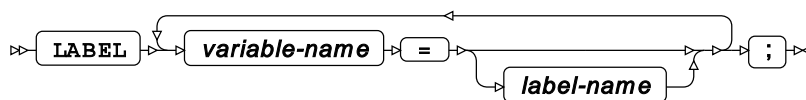


INFORMAT

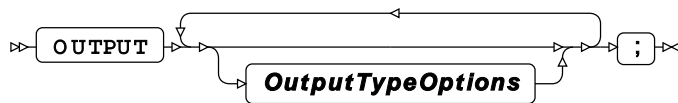


ⁱ See *Variable Lists* [↗](#) (page 32).

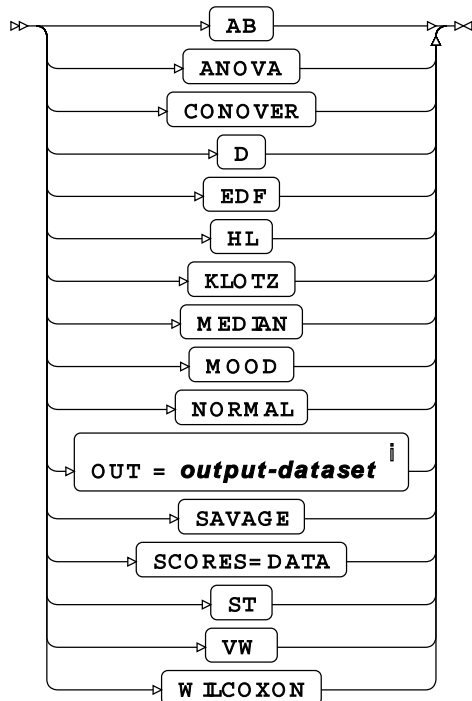
LABEL



OUTPUT

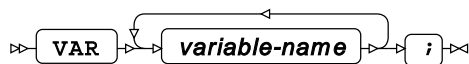


OutputTypeOptions

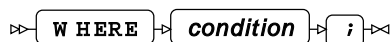


ⁱ See [Input dataset](#) (page 16).

VAR



WHERE

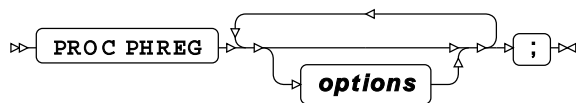


PHREG procedure

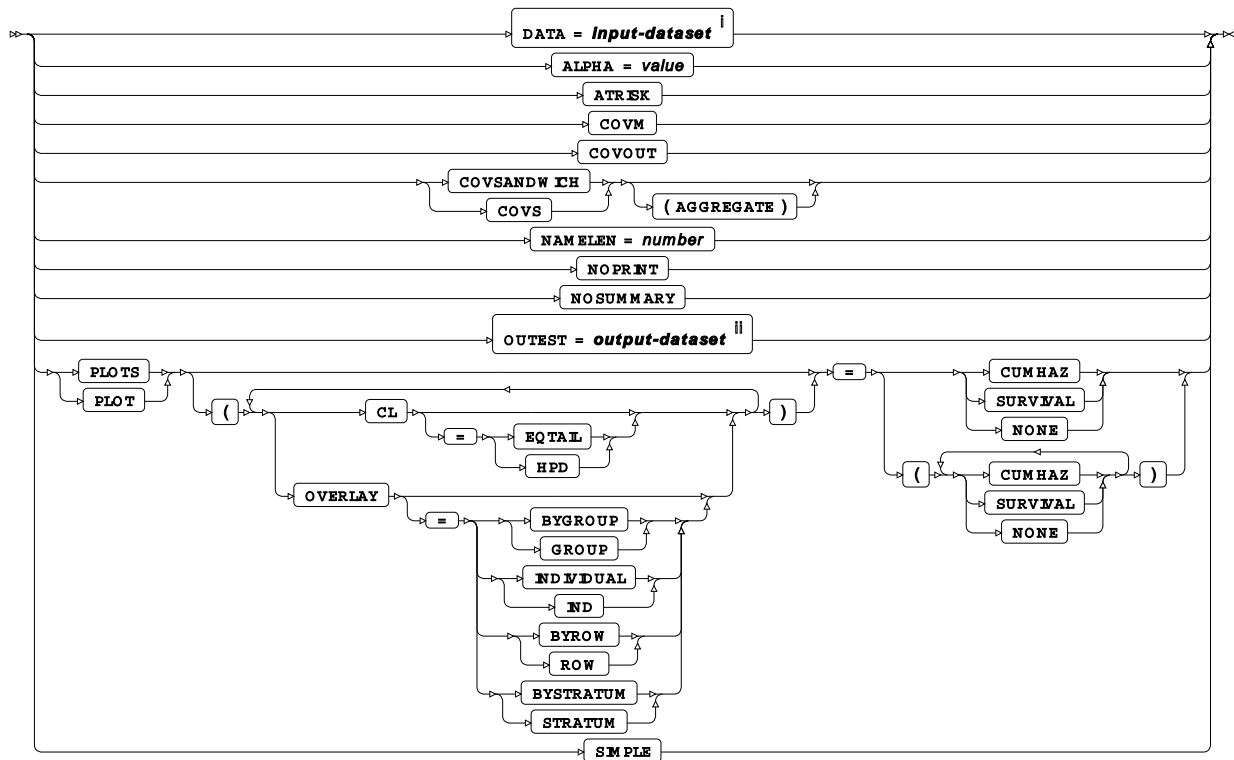
Supported statements

- *PROC PHREG* [↗](#) (page 3001)
- *ATTRIB* [↗](#) (page 3002)
- *BASELINE* [↗](#) (page 3003)
- *BY* [↗](#) (page 3004)
- *CLASS* [↗](#) (page 3004)
- *ESTIMATE* [↗](#) (page 3005)
- *FORMAT* [↗](#) (page 3006)
- *FREQ* [↗](#) (page 3007)
- *ID* [↗](#) (page 3007)
- *INFORMAT* [↗](#) (page 3007)
- *LABEL* [↗](#) (page 3007)
- *MODEL* [↗](#) (page 3007)
- *OUTPUT* [↗](#) (page 3010)
- *STRATA* [↗](#) (page 3011)
- *TEST* [↗](#) (page 3011)
- *WEIGHT* [↗](#) (page 3011)
- *WHERE* [↗](#) (page 3011)

PROC PHREG



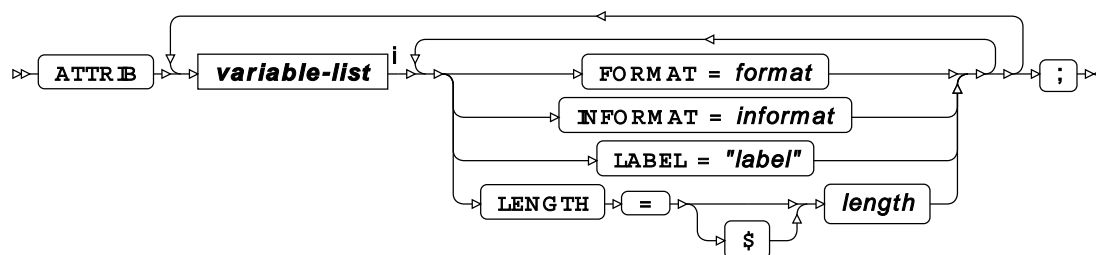
options



ⁱ See *Input dataset* [↗](#) (page 16).

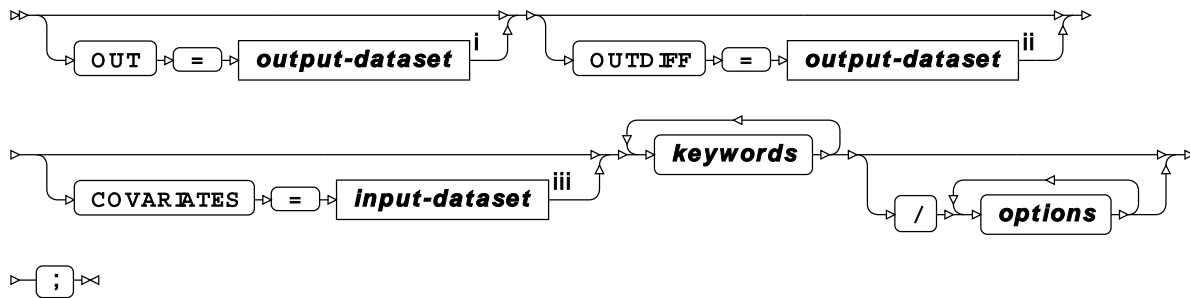
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BASELINE

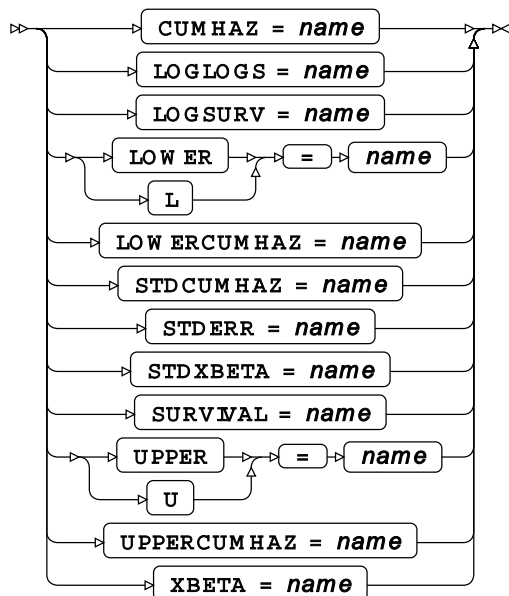


ⁱ See *Input dataset* [↗](#) (page 16).

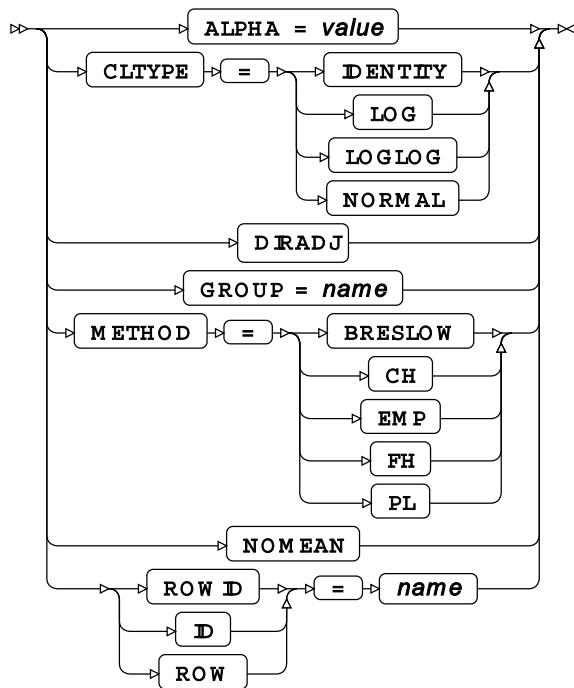
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

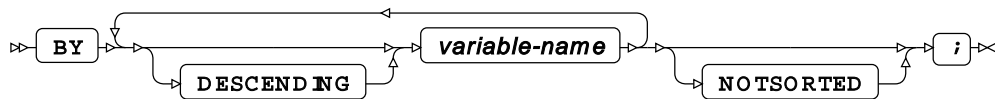
keywords



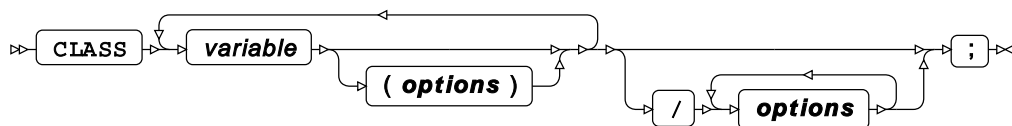
options



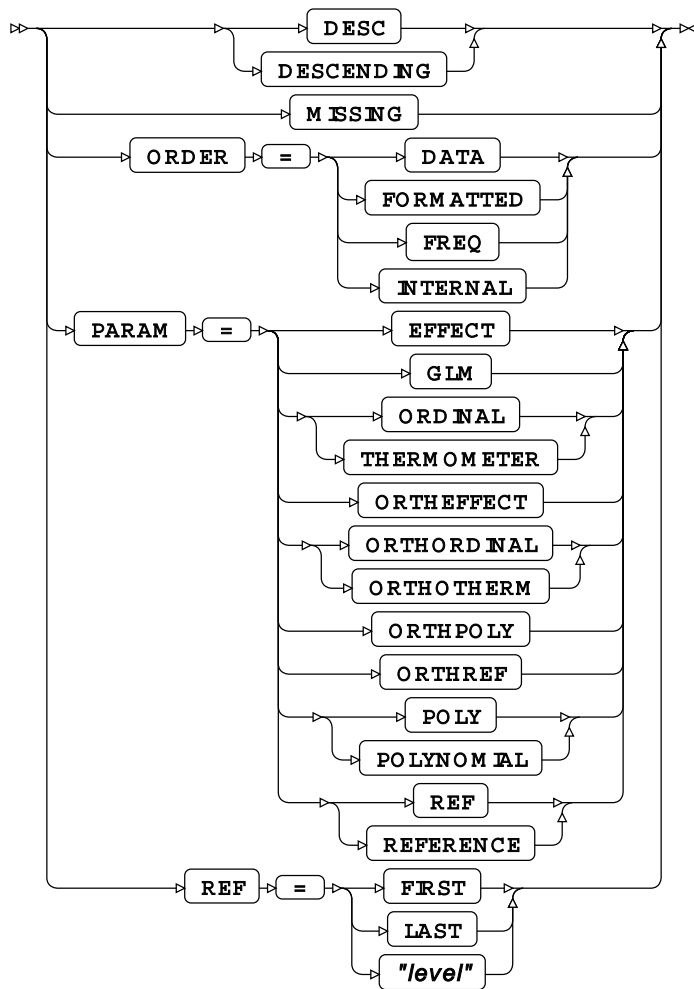
BY



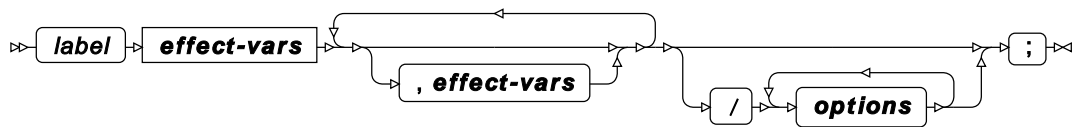
CLASS



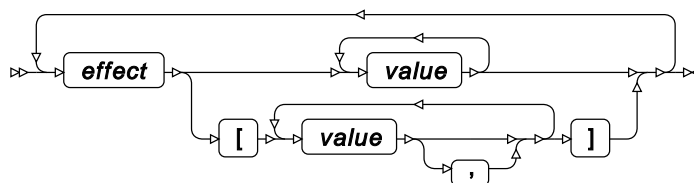
options



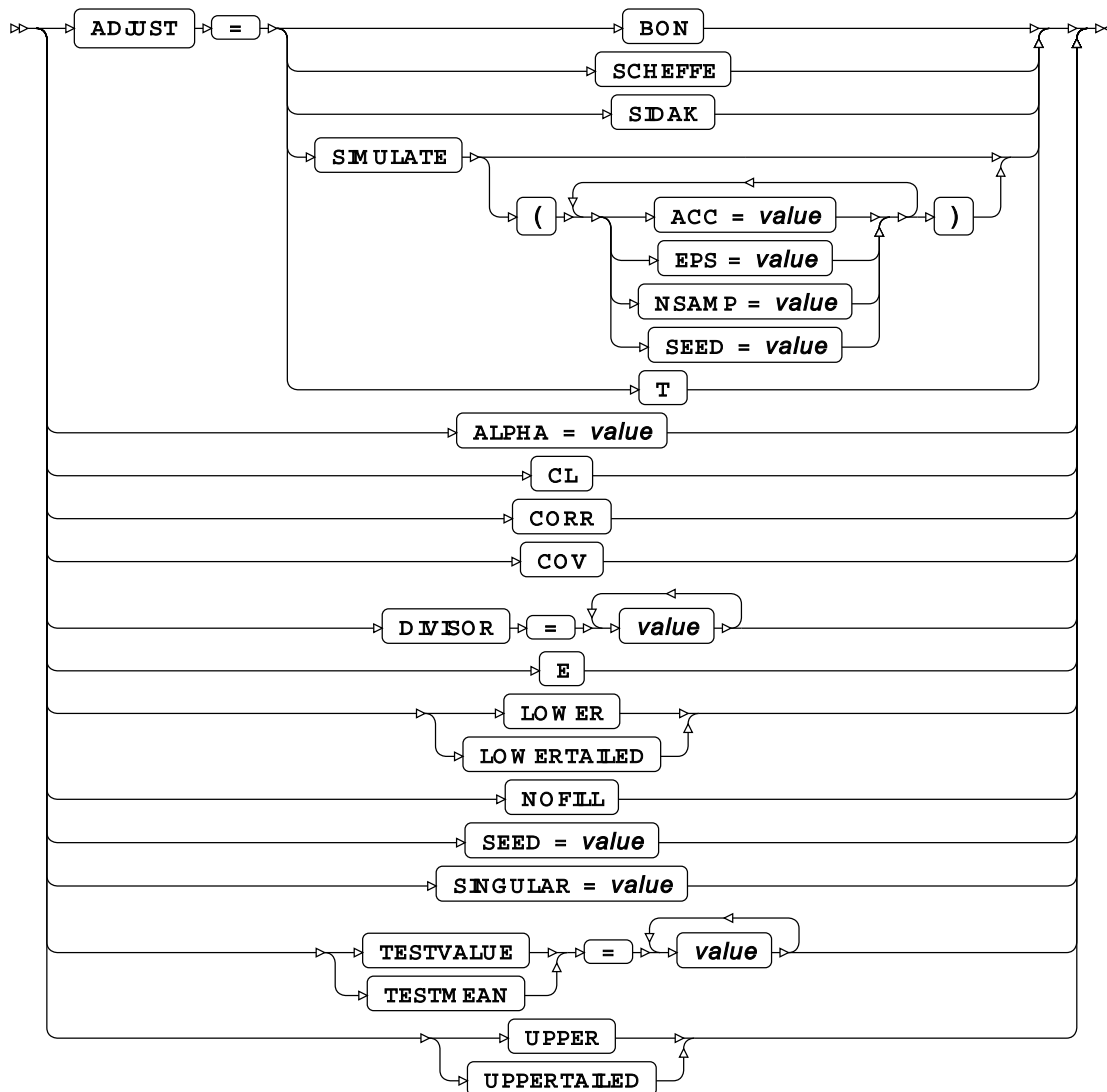
ESTIMATE



effect-vars



options

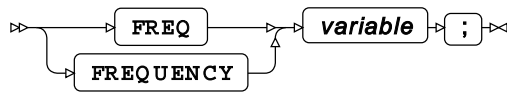


FORMAT

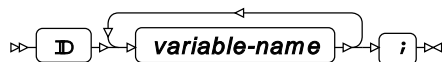


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

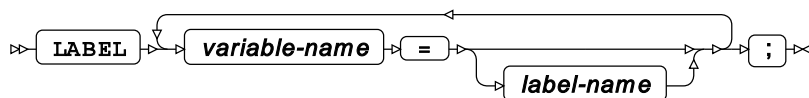


INFORMAT

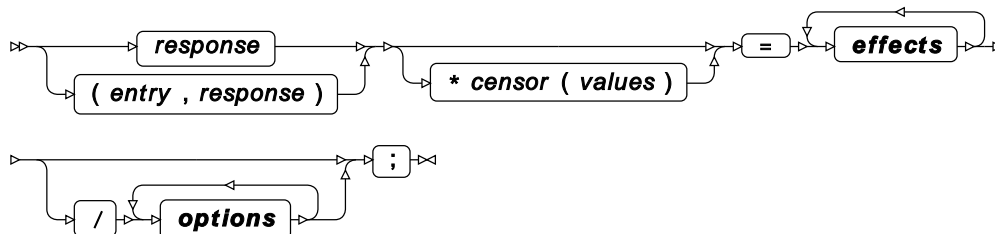


ⁱ See [Variable Lists](#) (page 32).

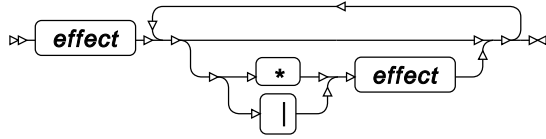
LABEL



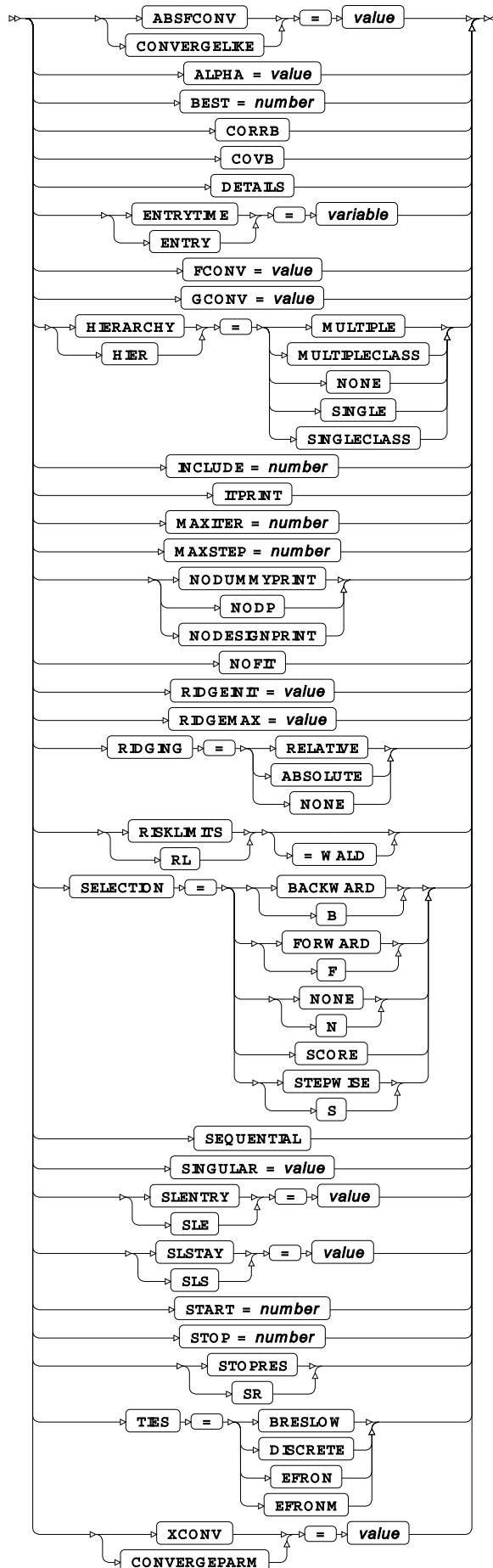
MODEL



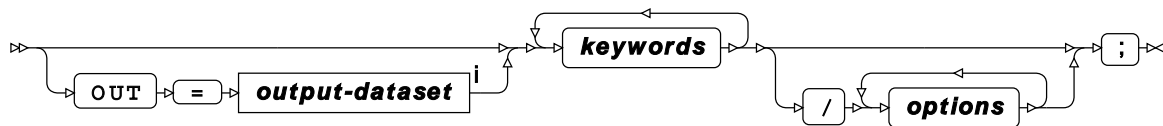
effects



options

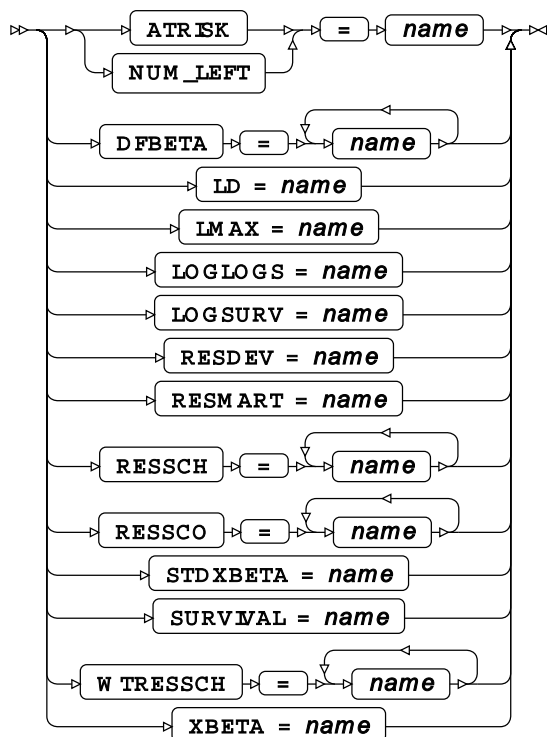


OUTPUT

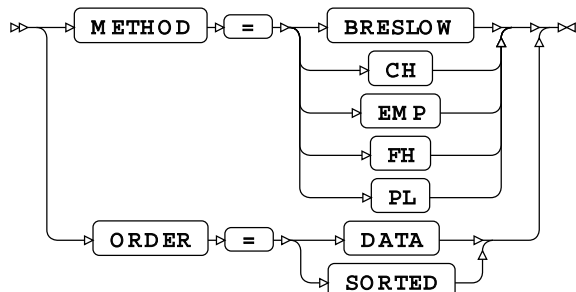


ⁱ See *Input dataset* [↗](#) (page 16).

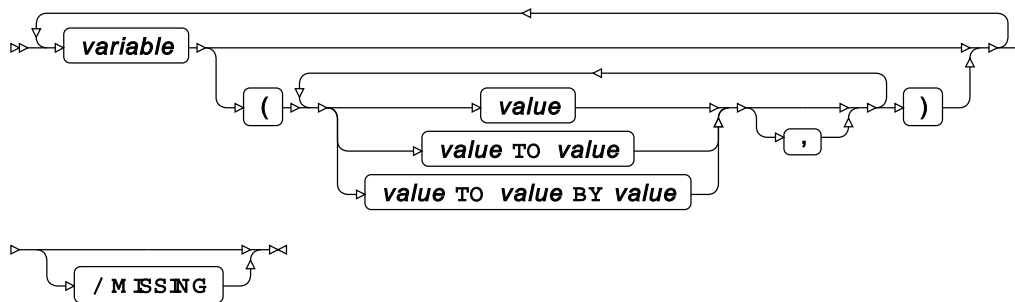
keywords



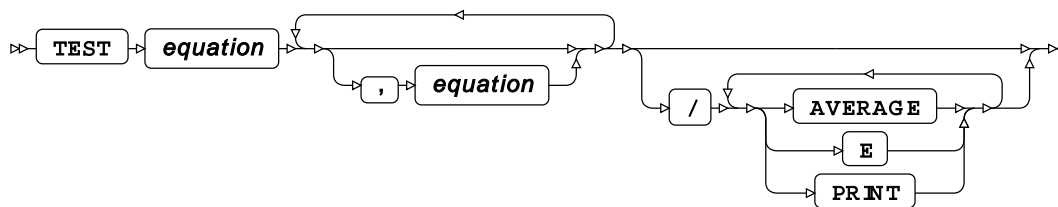
options



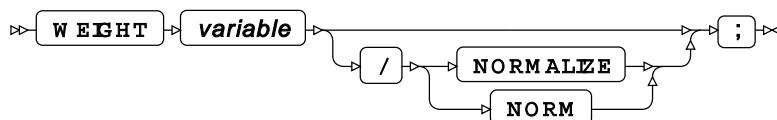
STRATA



TEST



WEIGHT



WHERE



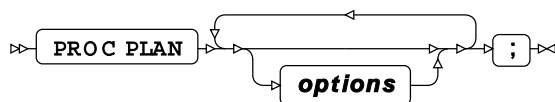
PLAN procedure

Supported statements

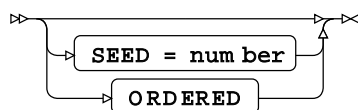
- *PROC PLAN* [↗](#) (page 3012)

- *ATTRIB* [↗](#) (page 3012)
- *FACTORS* [↗](#) (page 3013)
- *FORMAT* [↗](#) (page 3013)
- *INFORMAT* [↗](#) (page 3013)
- *LABEL* [↗](#) (page 3014)
- *OUTPUT* [↗](#) (page 3014)
- *TREATMENTS* [↗](#) (page 3014)
- *WHERE* [↗](#) (page 3014)

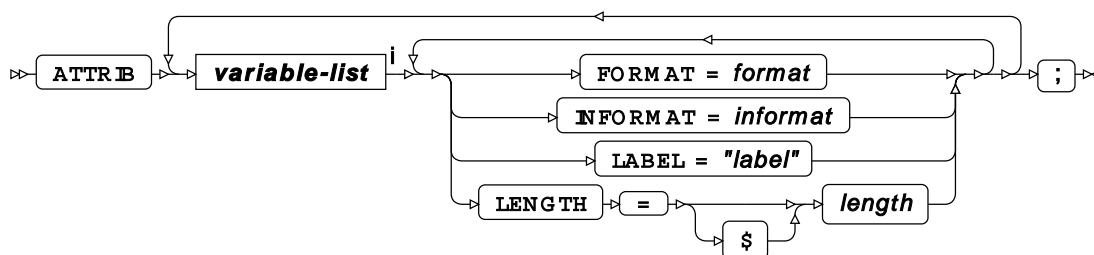
PROC PLAN



options

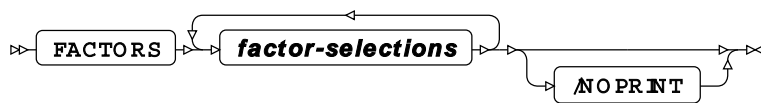


ATTRIB

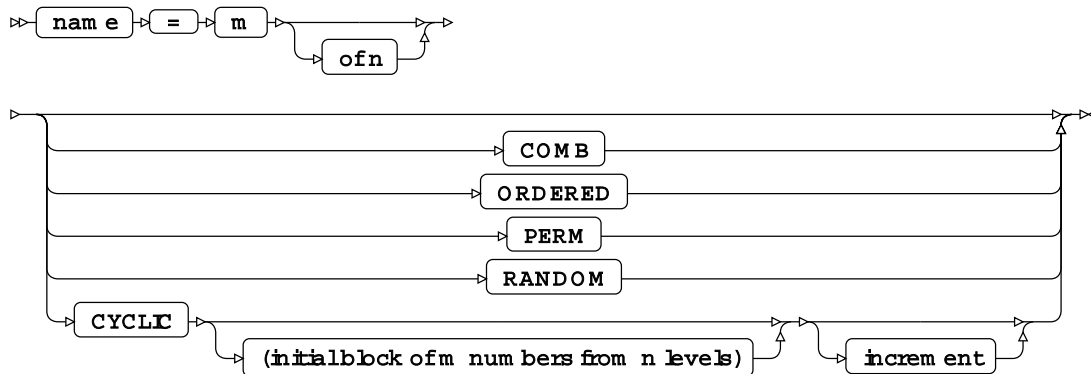


ⁱ See *Variable Lists* [↗](#) (page 32).

FACTORS



factor-selections



FORMAT



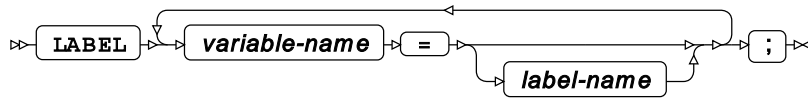
ⁱ See [Variable Lists](#) (page 32).

INFORMAT

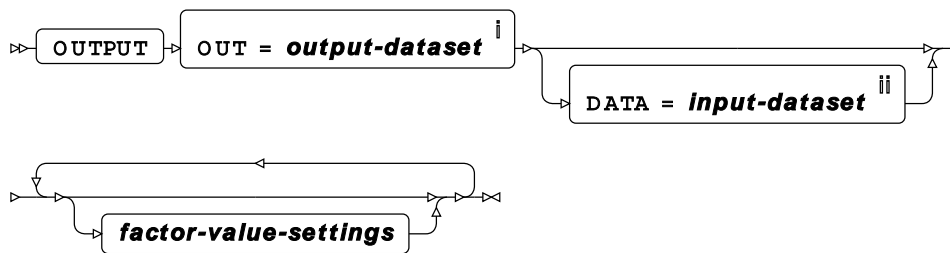


ⁱ See [Variable Lists](#) (page 32).

LABEL



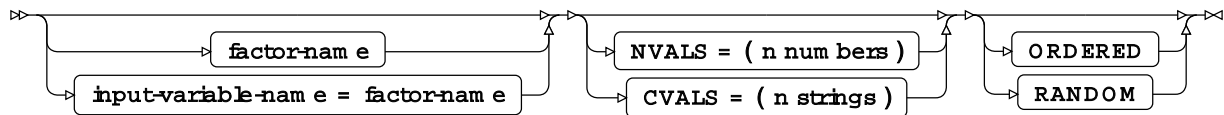
OUTPUT



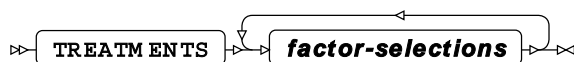
ⁱ See *Output dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

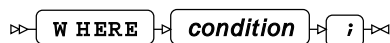
factor-value-settings



TREATMENTS



WHERE

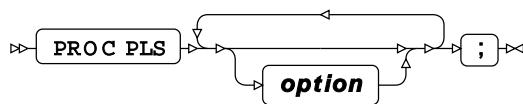


PLS procedure

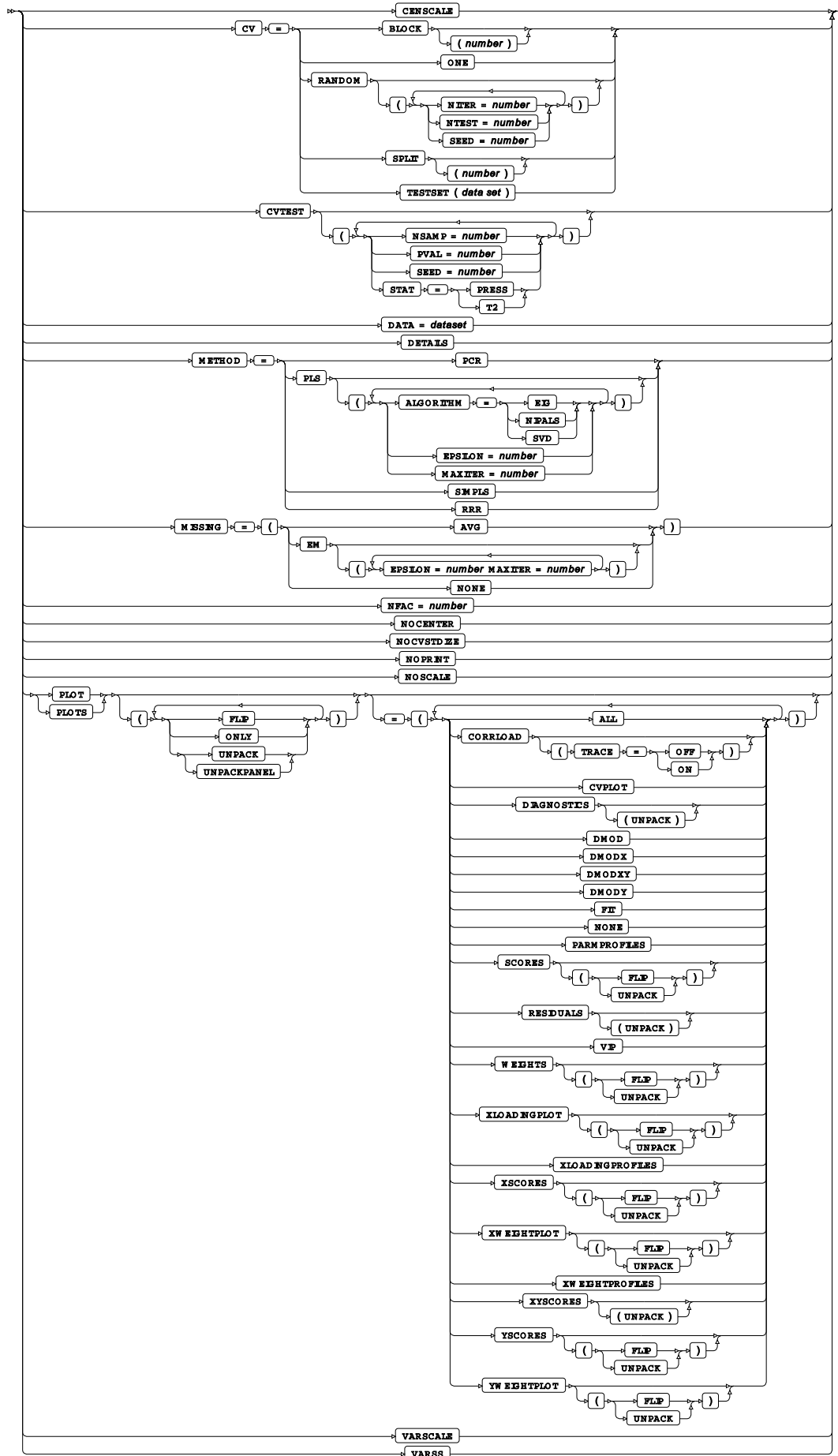
Supported statements

- *PROC PLS* [↗](#) (page 3015)
- *ATTRIB* [↗](#) (page 3017)
- *BY* [↗](#) (page 3017)
- *CLASS* [↗](#) (page 3017)
- *FORMAT* [↗](#) (page 3017)
- *ID* [↗](#) (page 3017)
- *INFORMAT* [↗](#) (page 3018)
- *LABEL* [↗](#) (page 3018)
- *MODEL* [↗](#) (page 3018)
- *OUTPUT* [↗](#) (page 3019)
- *WHERE* [↗](#) (page 3019)

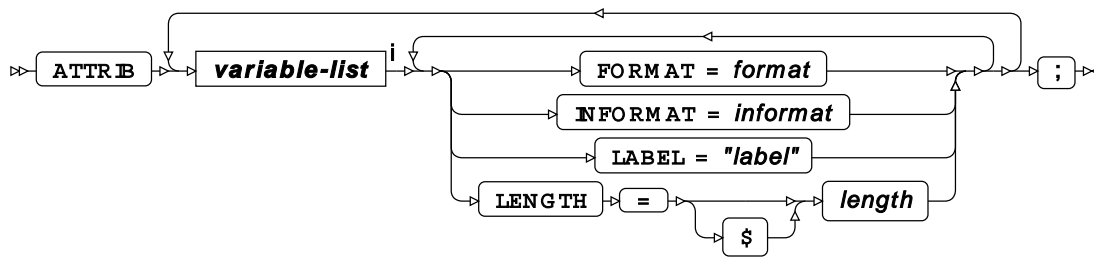
PROC PLS



option

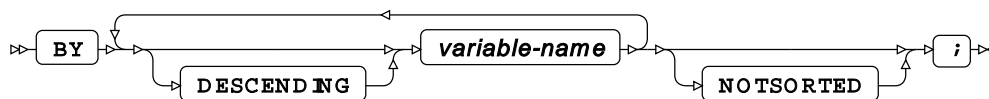


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY



CLASS

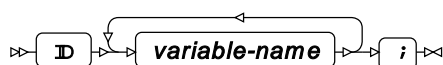


FORMAT

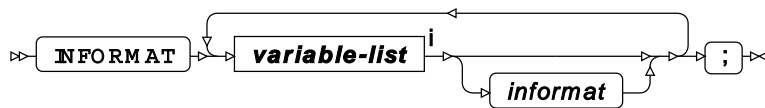


ⁱ See *Variable Lists* [↗](#) (page 32).

ID

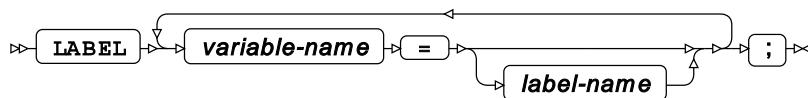


INFORMAT

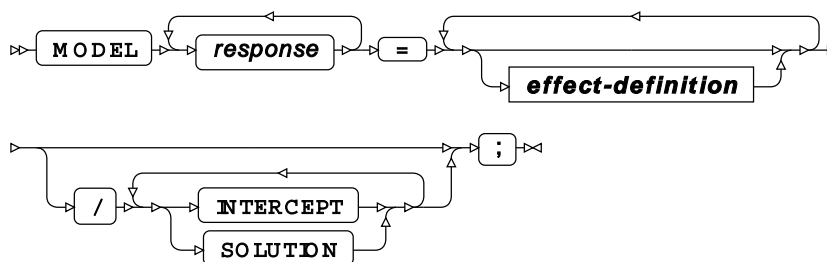


ⁱ See *Variable Lists* [↗](#) (page 32).

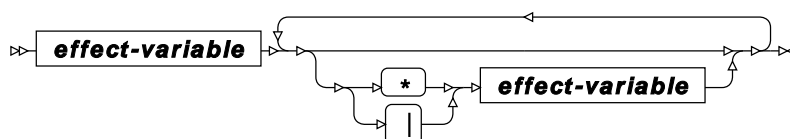
LABEL



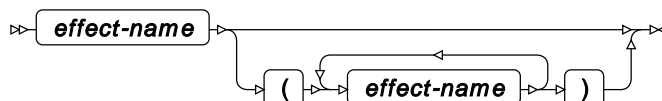
MODEL



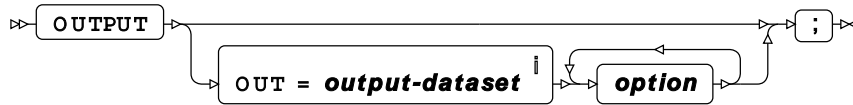
effect-definition



effect-variable

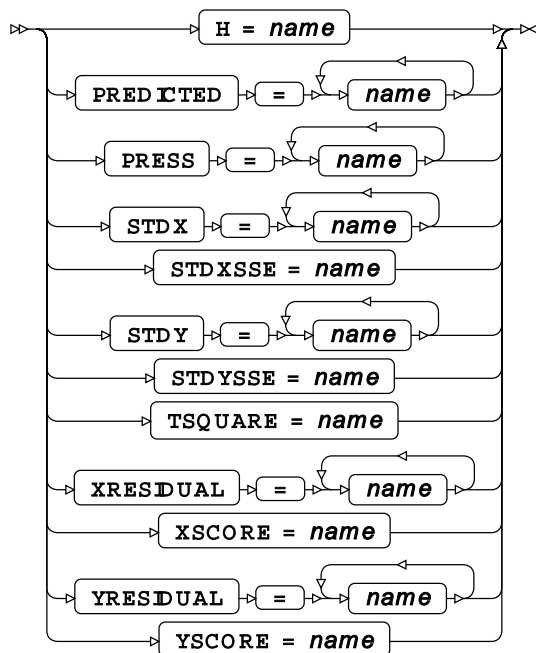


OUTPUT

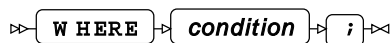


ⁱ See *Output dataset* [↗](#) (page 16).

option



WHERE



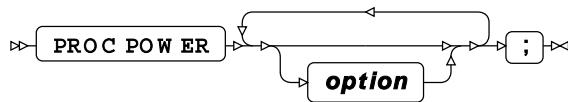
POWER procedure

Supported statements

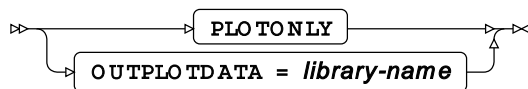
- *PROC POWER* [↗](#) (page 3020)
- *ATTRIB* [↗](#) (page 3021)
- *FORMAT* [↗](#) (page 3021)

- *INFORMAT* [↗](#) (page 3021)
- *LABEL* [↗](#) (page 3021)
- *LOGISTIC* [↗](#) (page 3022)
- *MULTREG* [↗](#) (page 3026)
- *ONECORR* [↗](#) (page 3029)
- *ONESAMPLEFREQ* [↗](#) (page 3032)
- *ONESAMPLEMEANS* [↗](#) (page 3036)
- *ONEWAYANOVA* [↗](#) (page 3040)
- *PAIREDFREQ* [↗](#) (page 3043)
- *PAIREDMEANS* [↗](#) (page 3046)
- *PLOT* [↗](#) (page 3050)
- *TWOSAMPLEFREQ* [↗](#) (page 3054)
- *TWOSAMPLEMEANS* [↗](#) (page 3058)
- *TWOSAMPLESURVIVAL* [↗](#) (page 3063)
- *TWOSAMPLEWILCOXON* [↗](#) (page 3068)
- *WHERE* [↗](#) (page 3071)

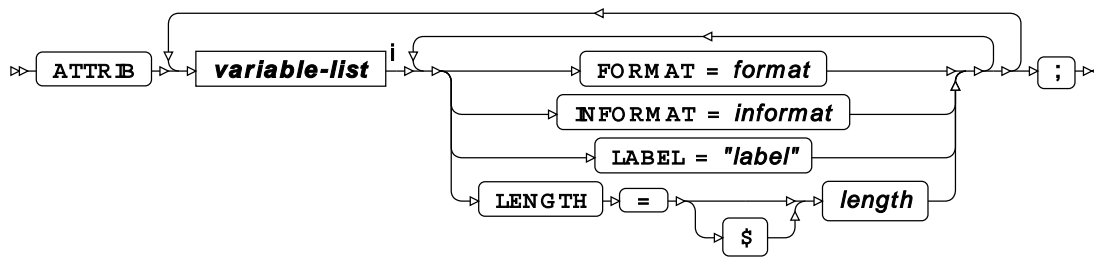
PROC POWER



option



ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

FORMAT



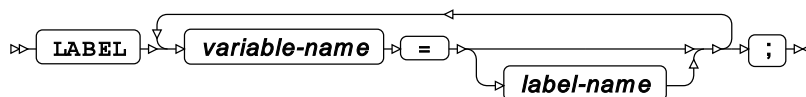
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

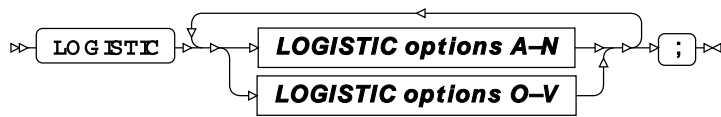


ⁱ See *Variable Lists* [↗](#) (page 32).

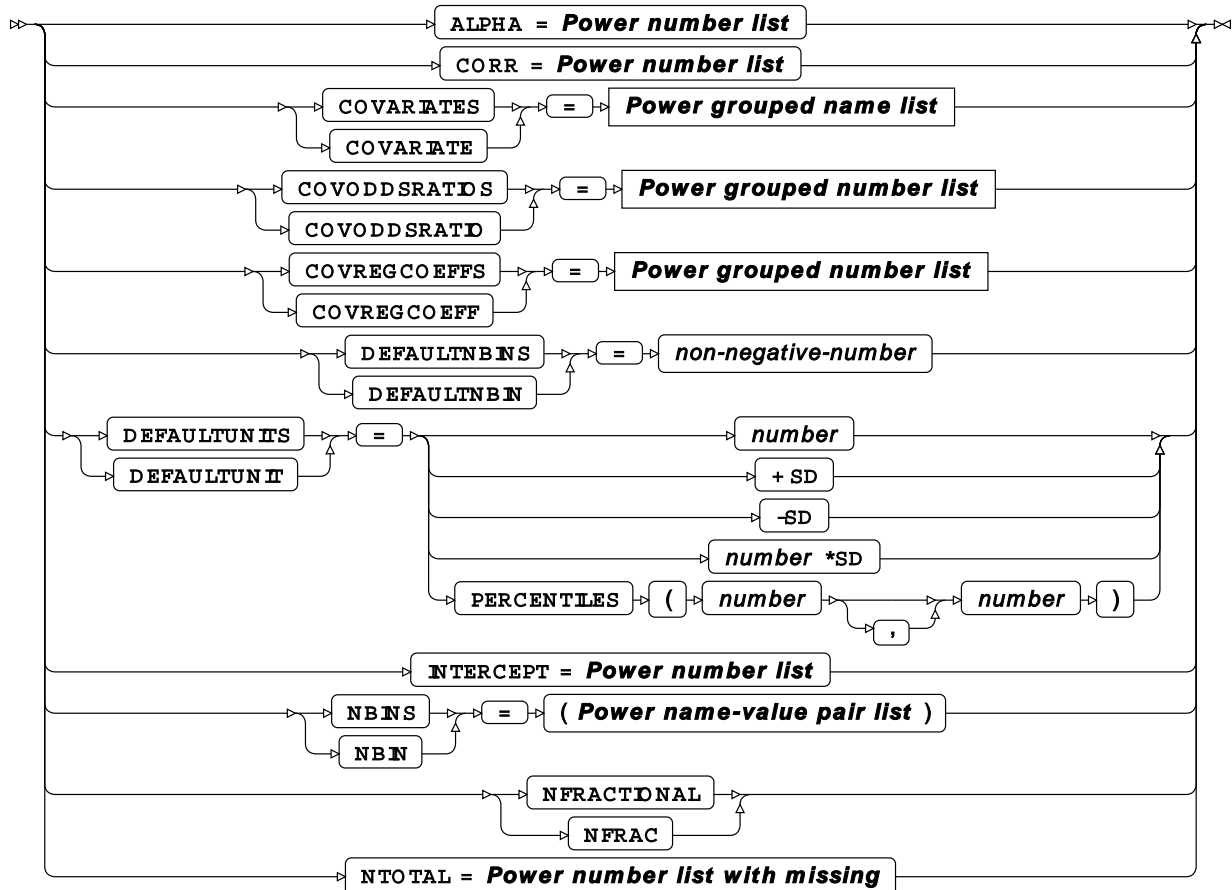
LABEL



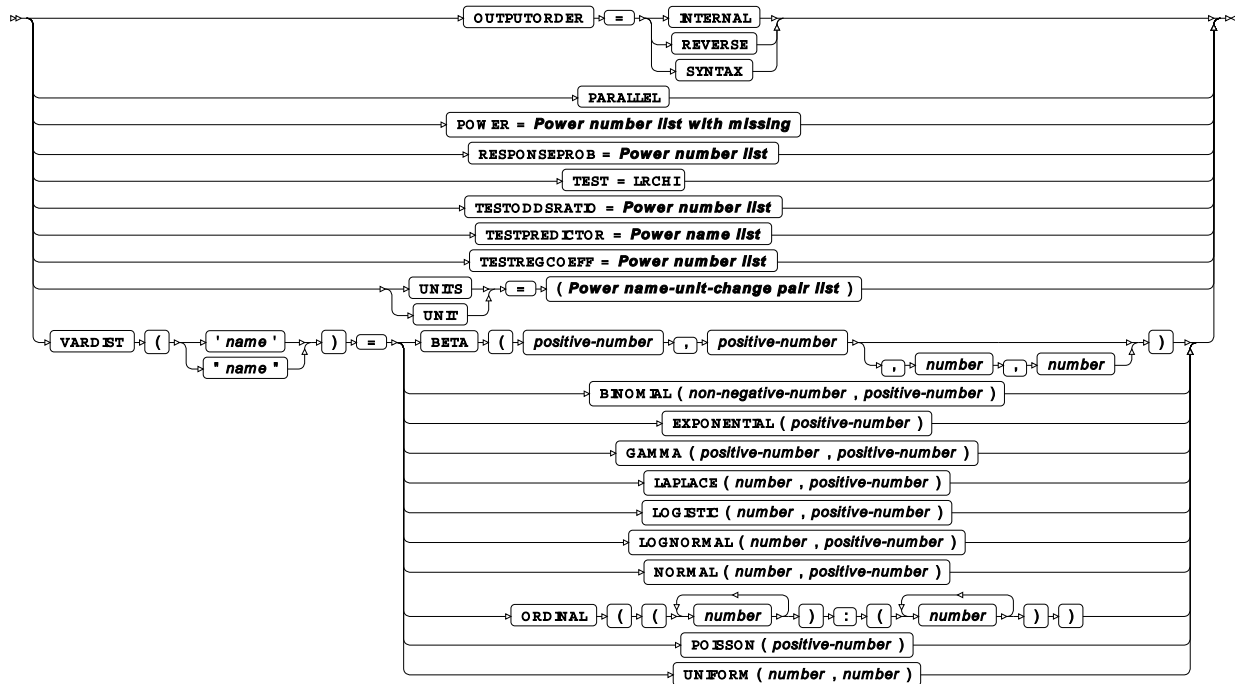
LOGISTIC



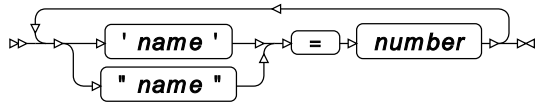
LOGISTIC options A–N



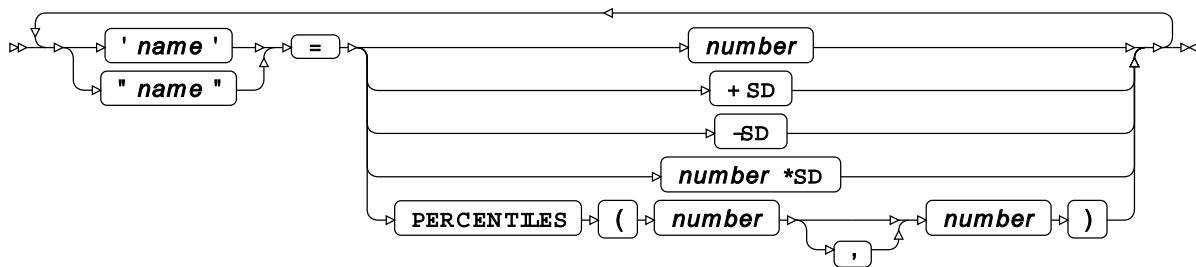
LOGISTIC options O–V



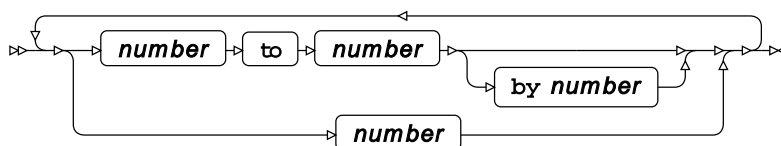
Power name-value pair list



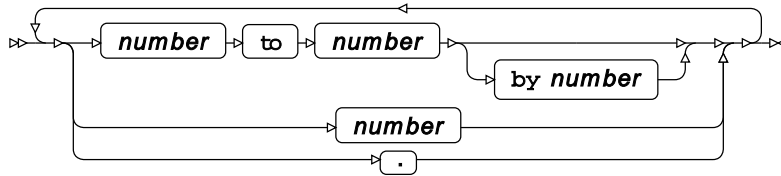
Power name-unit-change pair list



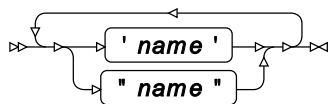
Power number list



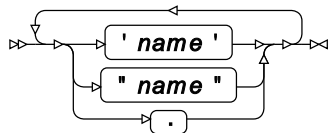
Power number list with missing



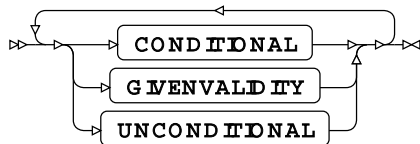
Power name list



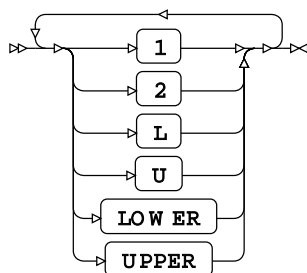
Power name list with missing



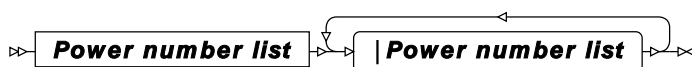
Power probtype list



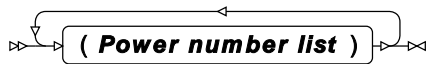
Power sides list



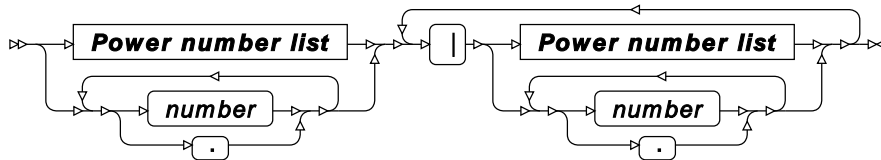
Power crossed number list



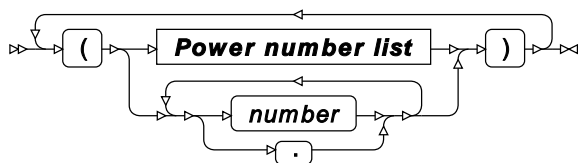
Power matched number list



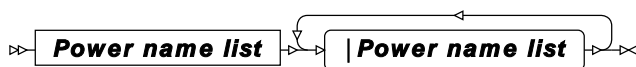
Power crossed number list



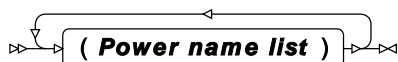
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



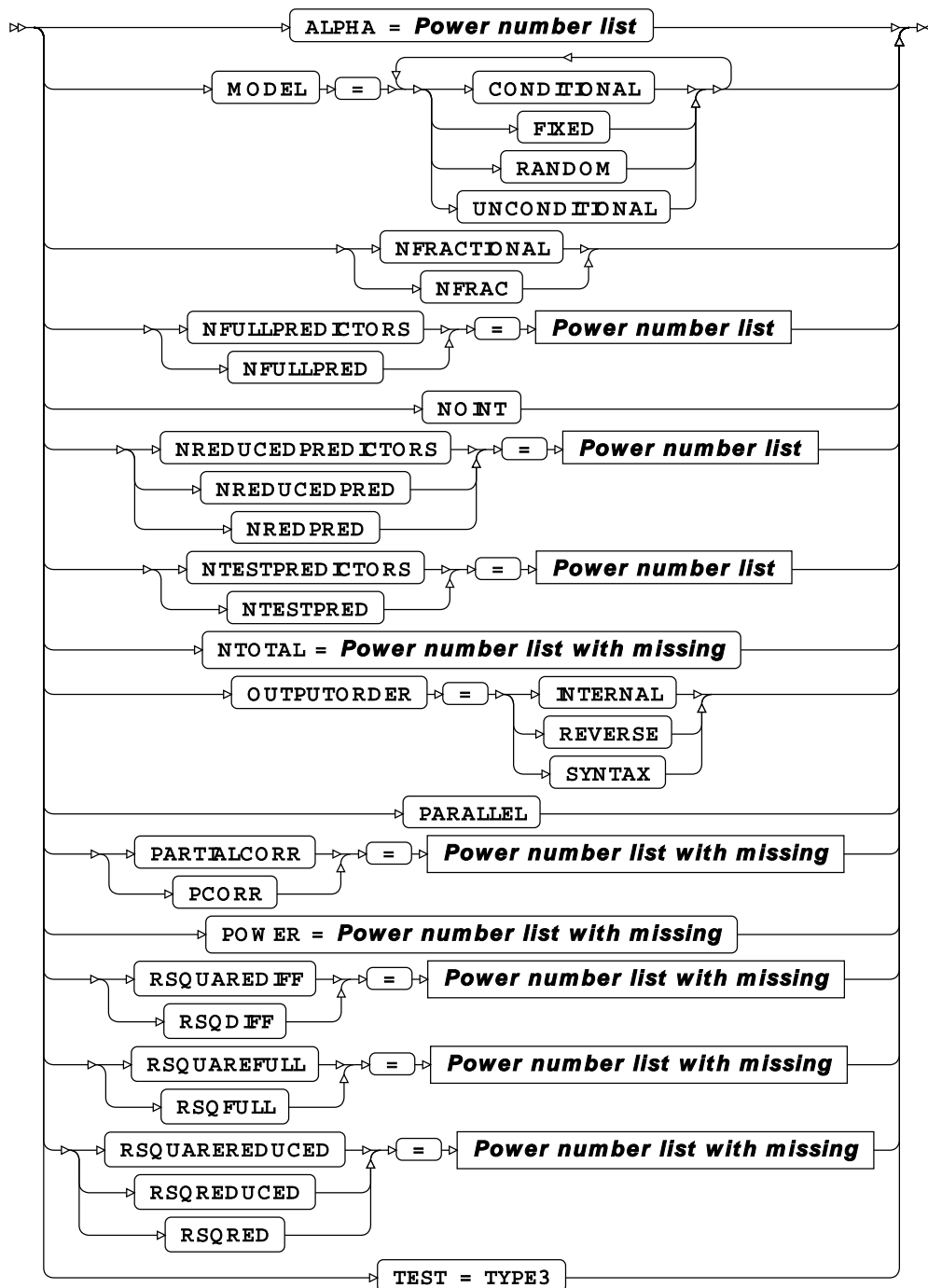
Power grouped name list



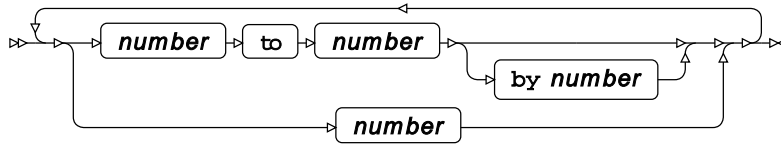
MULTREG



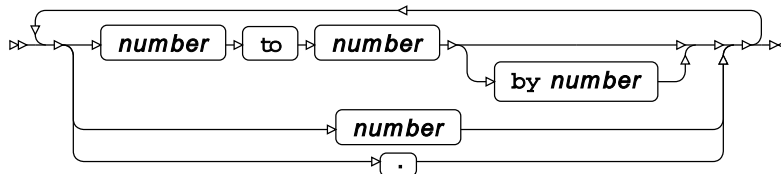
MULTREG options



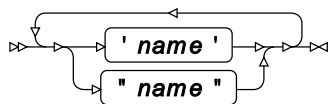
Power number list



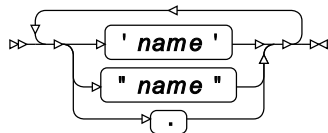
Power number list with missing



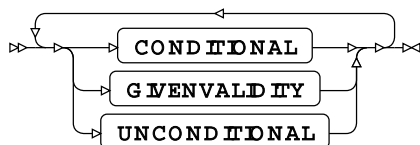
Power name list



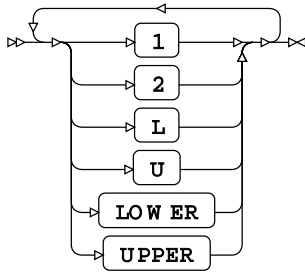
Power name list with missing



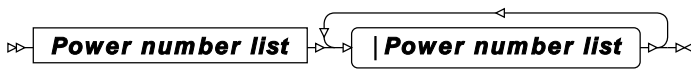
Power probtype list



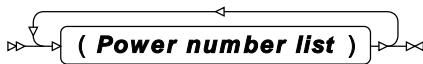
Power sides list



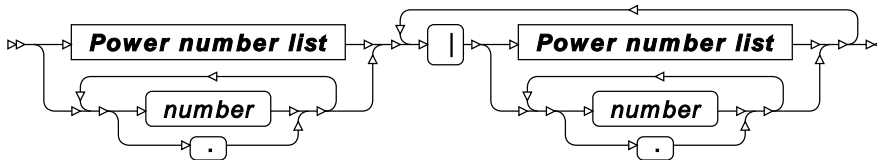
Power crossed number list



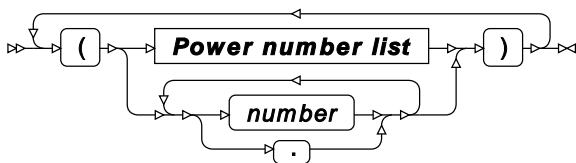
Power matched number list



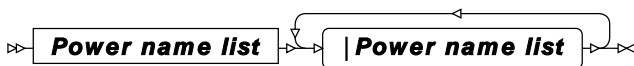
Power crossed number list



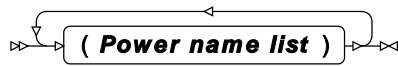
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



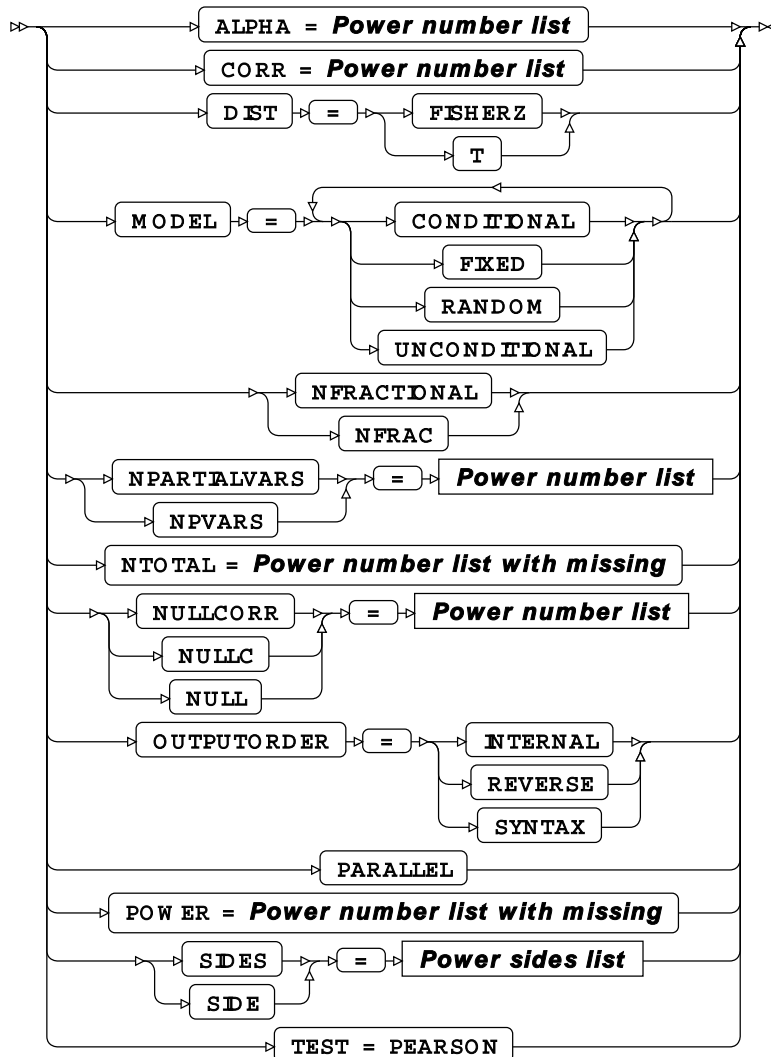
Power grouped name list



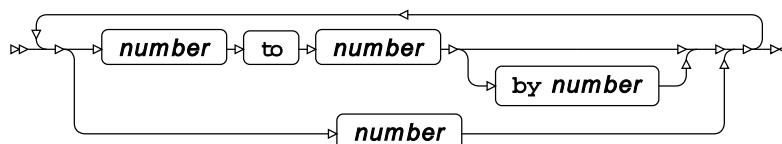
ONECORR



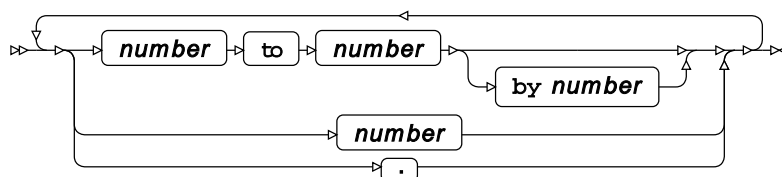
ONECORR options



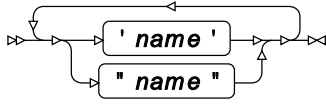
Power number list



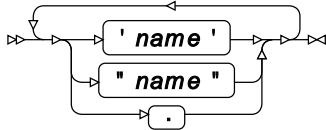
Power number list with missing



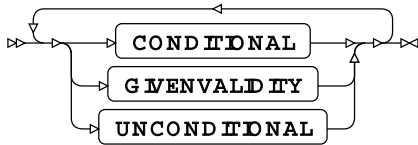
Power name list



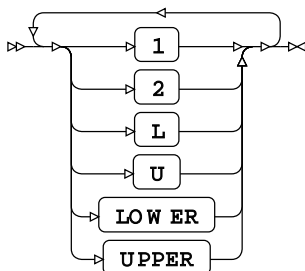
Power name list with missing



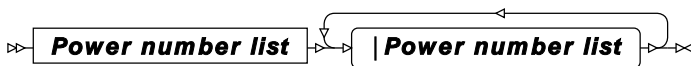
Power probtype list



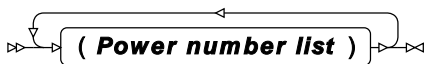
Power sides list



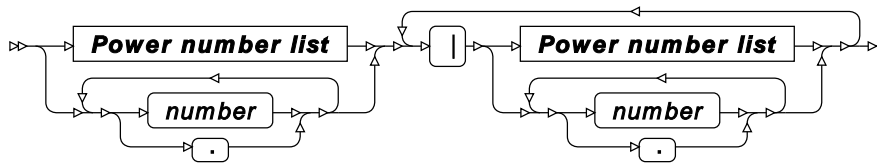
Power crossed number list



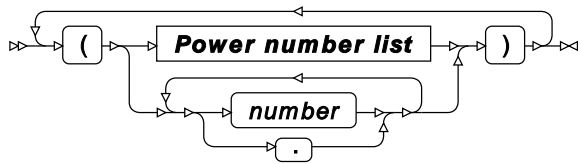
Power matched number list



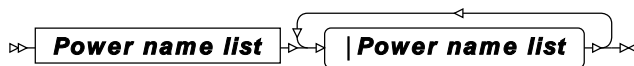
Power crossed number list



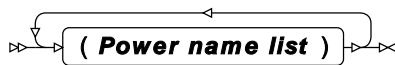
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



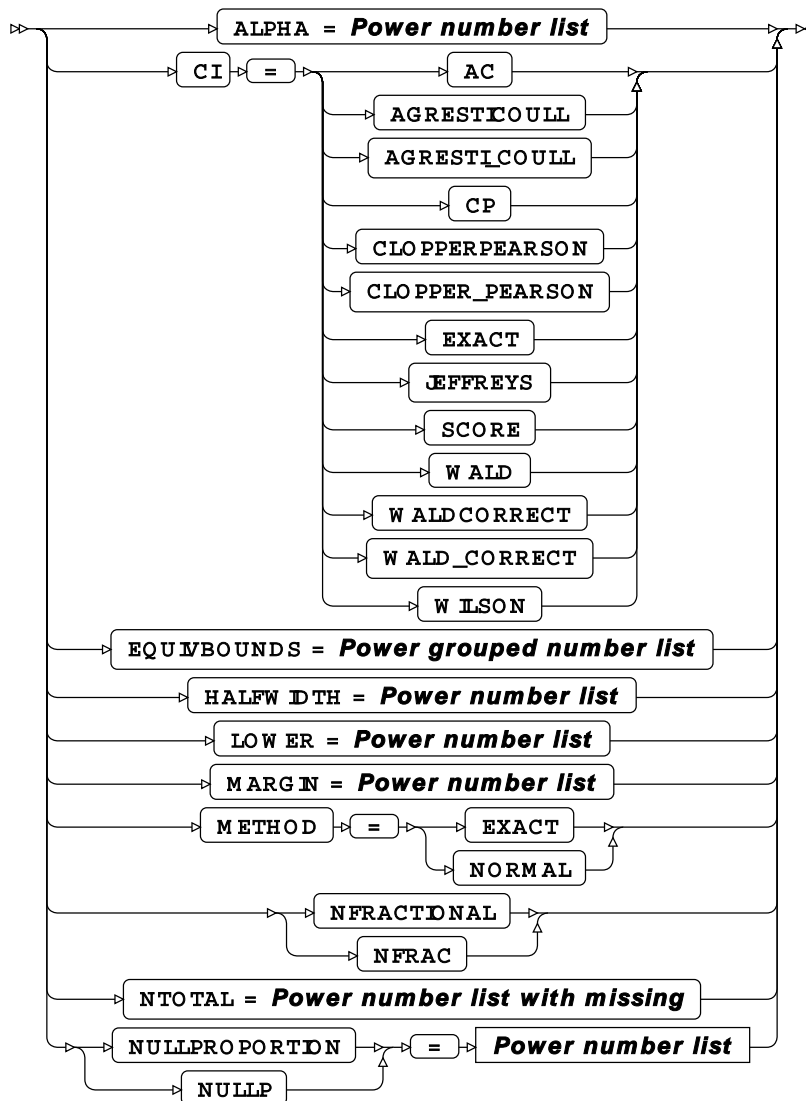
Power grouped name list



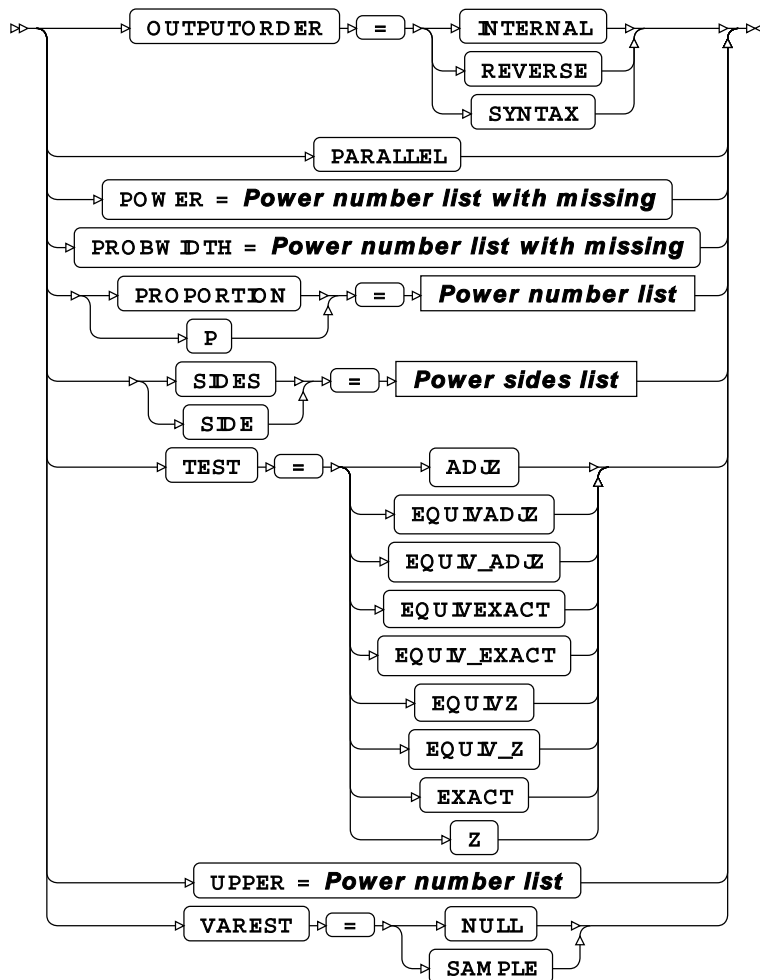
ONESAMPLEFREQ



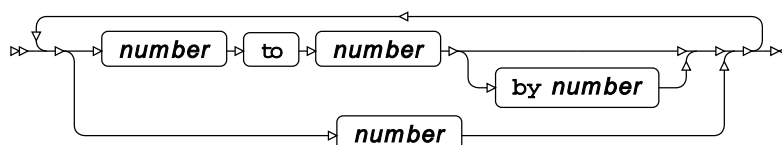
ONESAMPLEFREQ options A–N



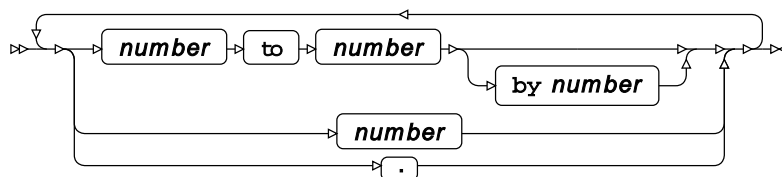
ONESAMPLEFREQ options O–V



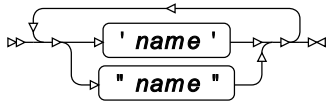
Power number list



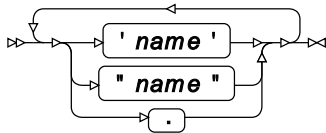
Power number list with missing



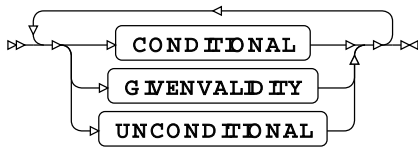
Power name list



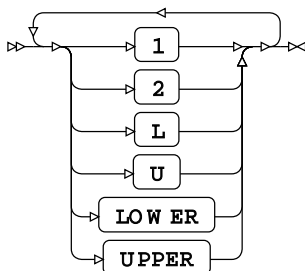
Power name list with missing



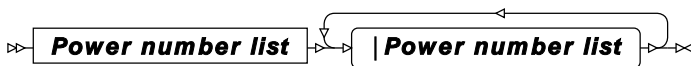
Power probtype list



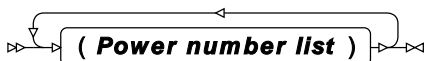
Power sides list



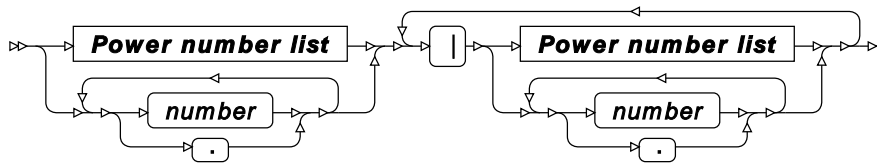
Power crossed number list



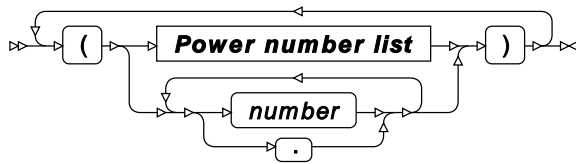
Power matched number list



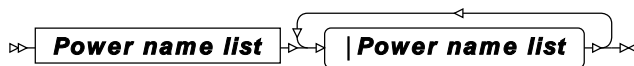
Power crossed number list



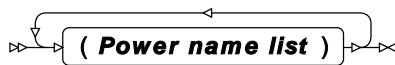
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



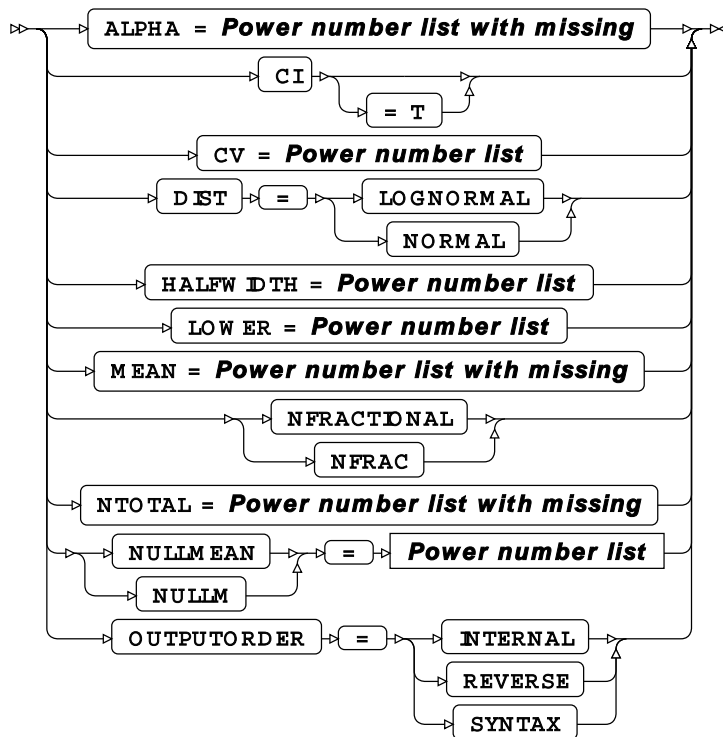
Power grouped name list



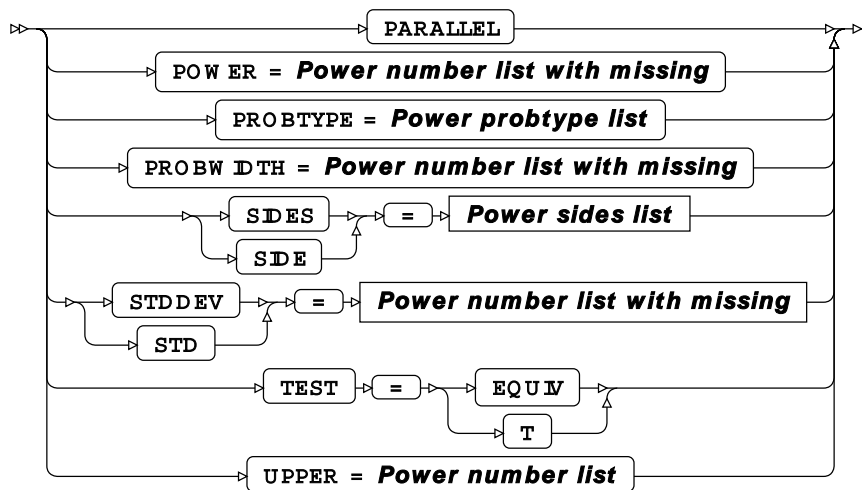
ONESAMPLEMEANS



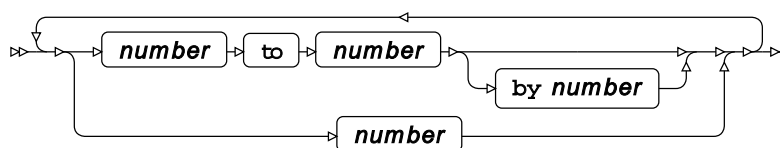
ONESAMPLEMEANS options A–O



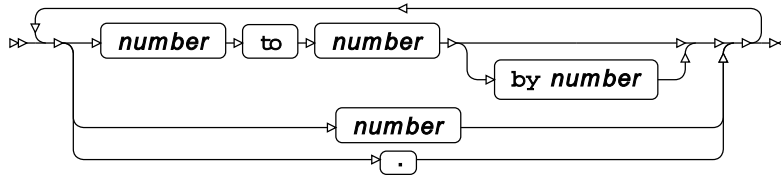
ONESAMPLEMEANS options P–V



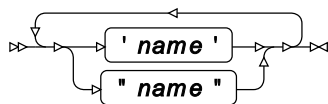
Power number list



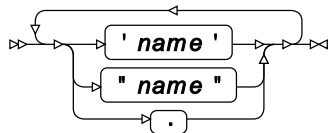
Power number list with missing



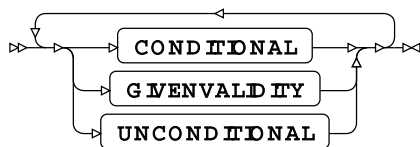
Power name list



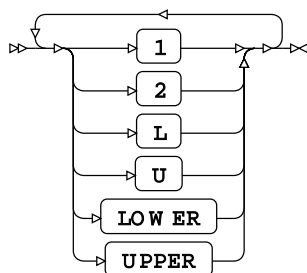
Power name list with missing



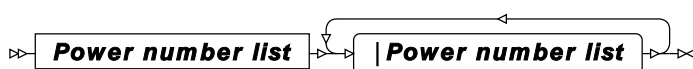
Power probtype list



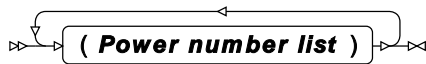
Power sides list



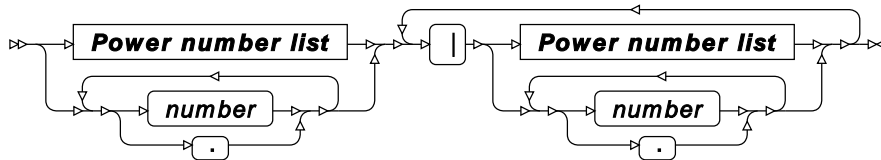
Power crossed number list



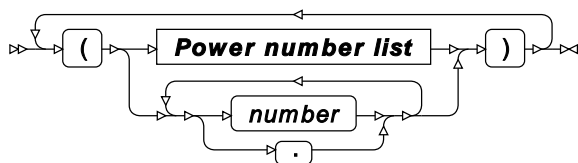
Power matched number list



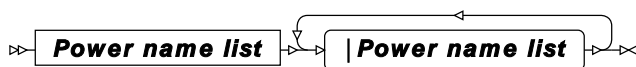
Power crossed number list



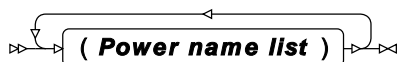
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



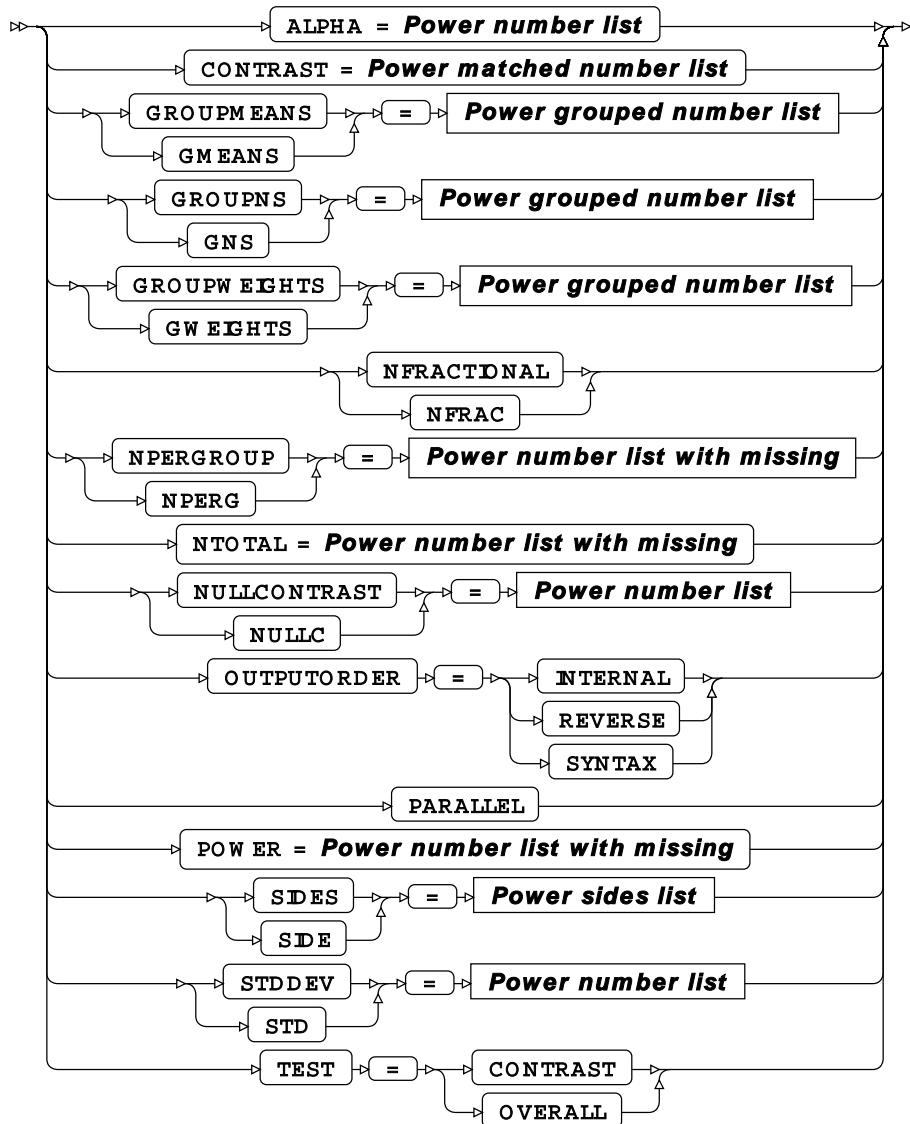
Power grouped name list



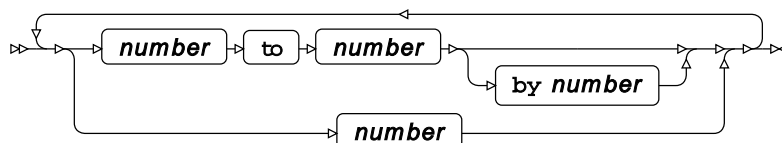
ONEWAYANOVA



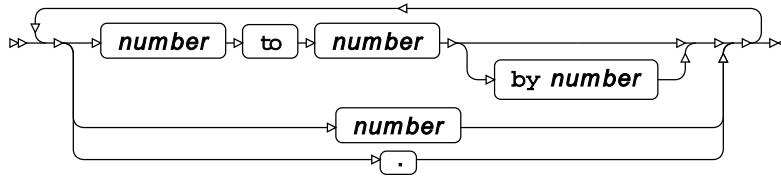
ONEWAYANOVA options



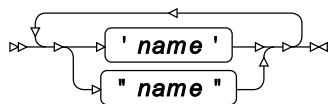
Power number list



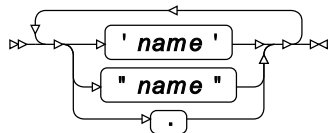
Power number list with missing



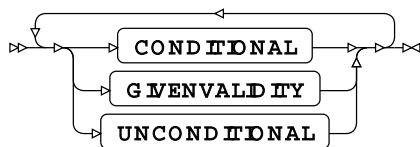
Power name list



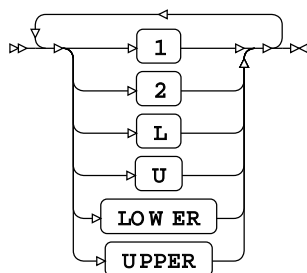
Power name list with missing



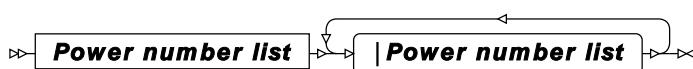
Power probtype list



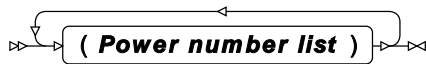
Power sides list



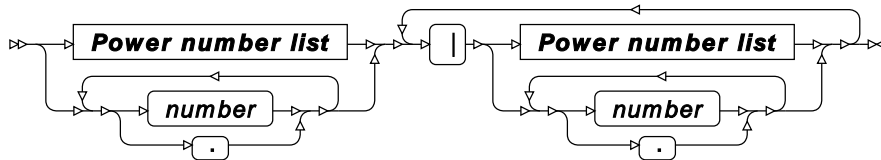
Power crossed number list



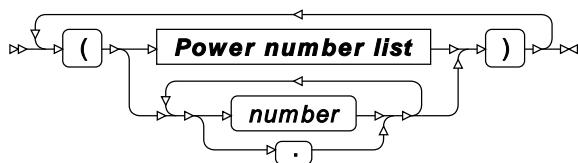
Power matched number list



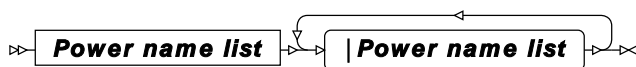
Power crossed number list



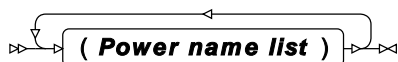
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



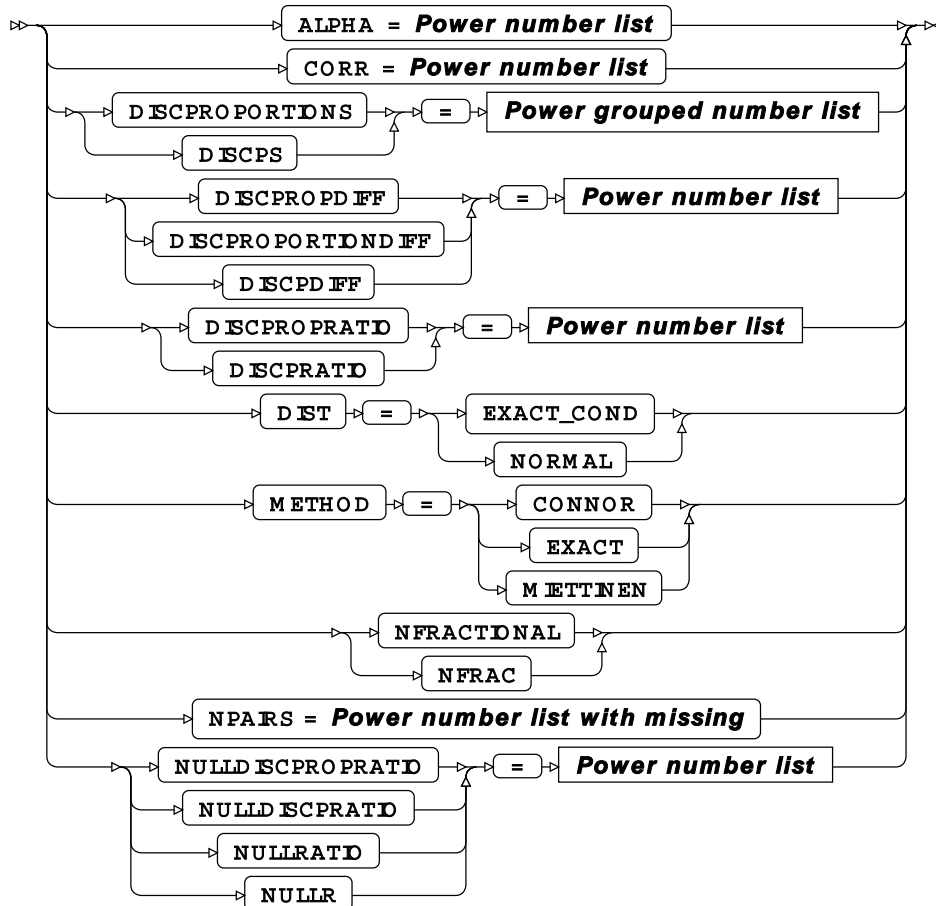
Power grouped name list



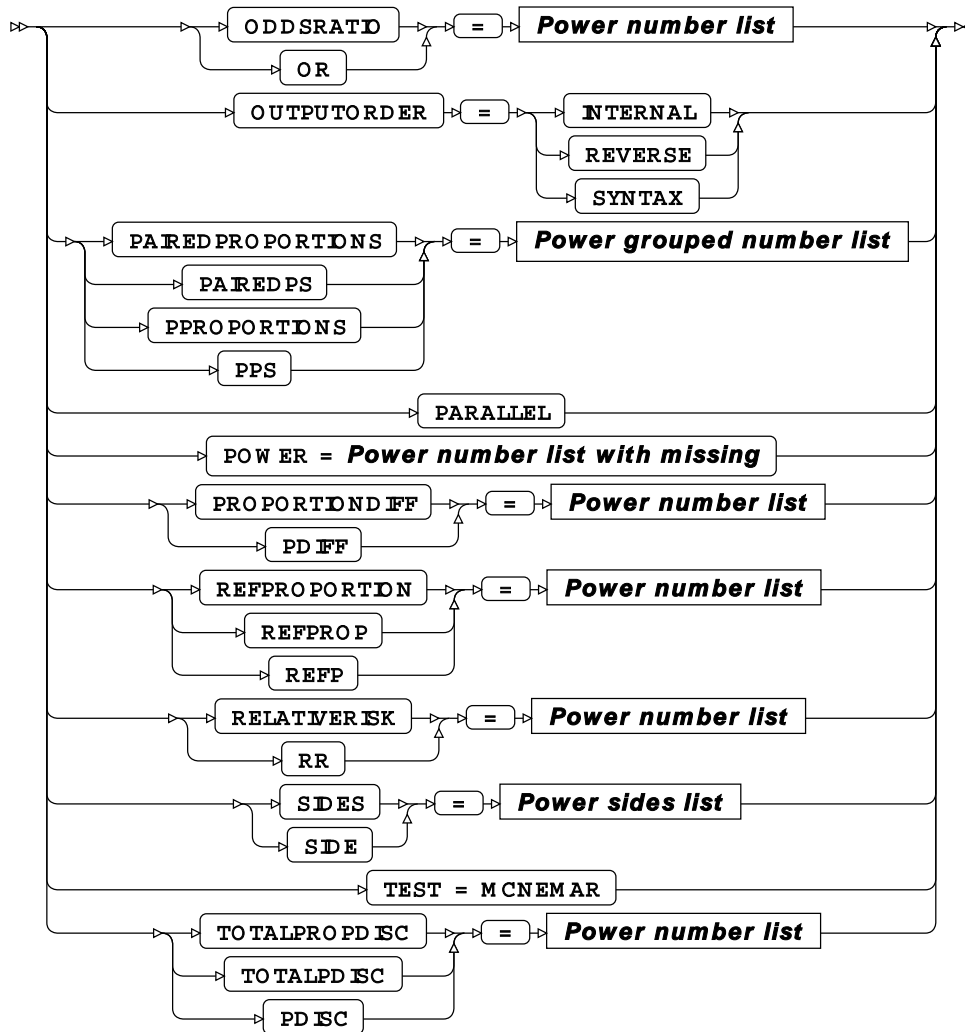
PAIREDFREQ



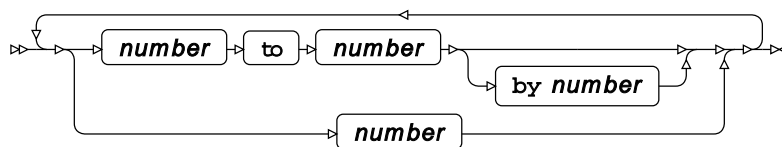
PAIREDFREQ options A-N



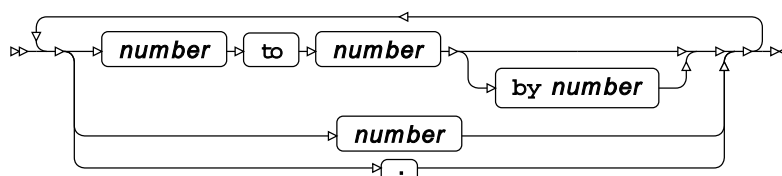
PAIREDFREQ options O–V



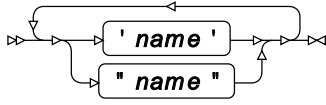
Power number list



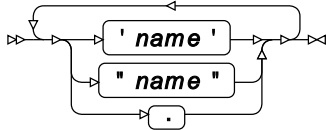
Power number list with missing



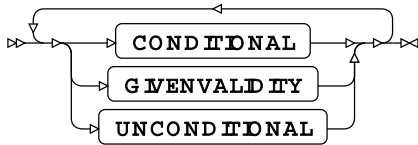
Power name list



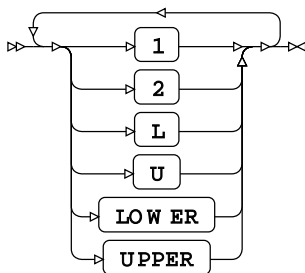
Power name list with missing



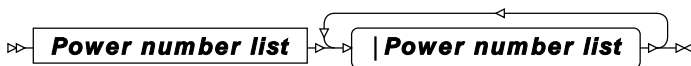
Power probtype list



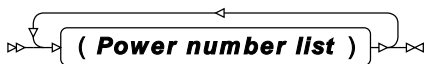
Power sides list



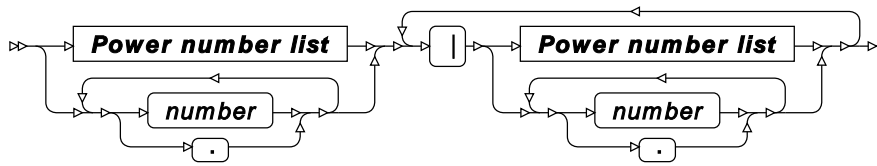
Power crossed number list



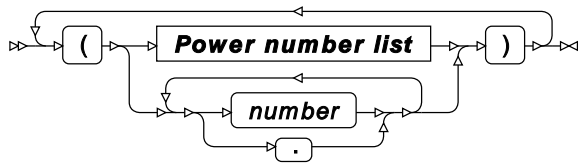
Power matched number list



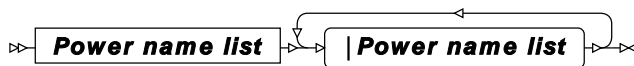
Power crossed number list



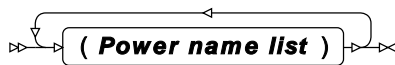
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



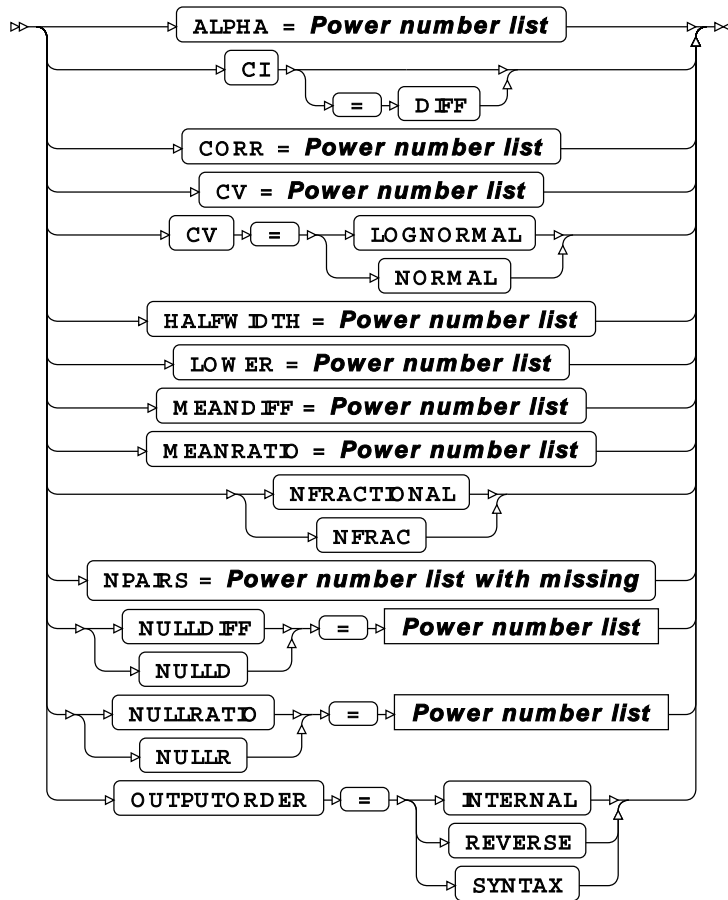
Power grouped name list



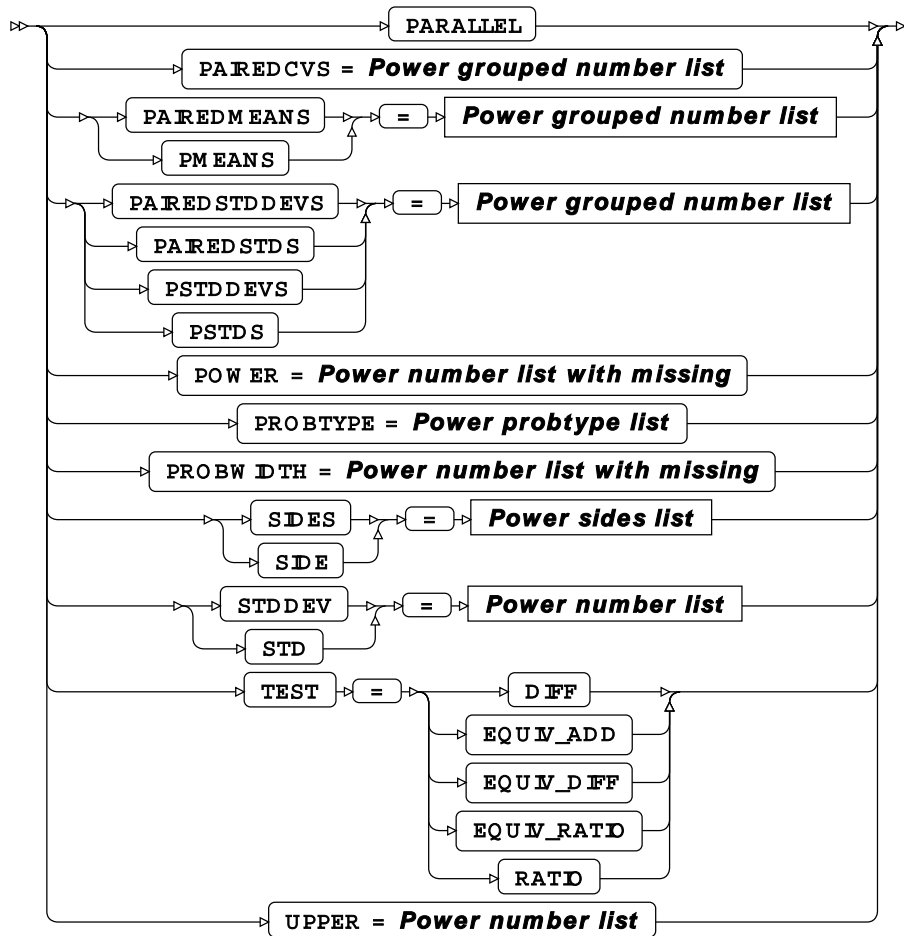
PAIREDMEANS



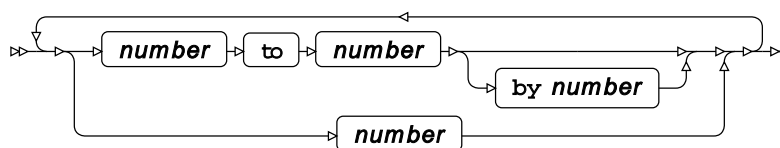
PAIREDMEANS options A–O



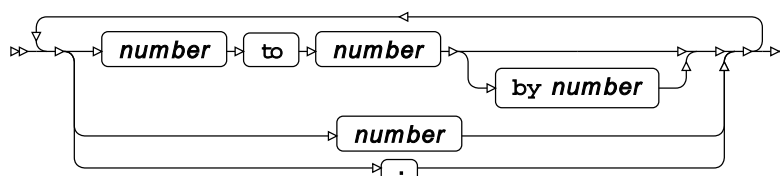
PAIREDMEANS options P–V



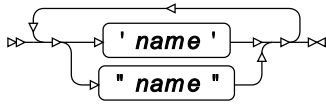
Power number list



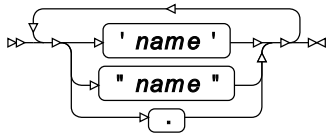
Power number list with missing



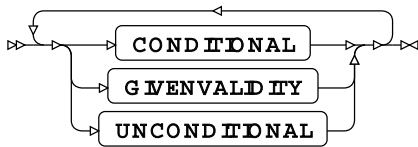
Power name list



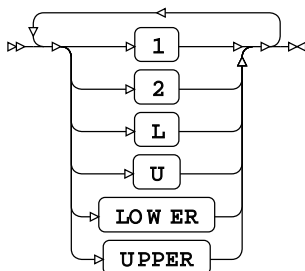
Power name list with missing



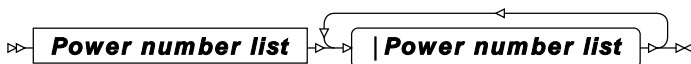
Power probtype list



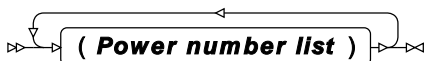
Power sides list



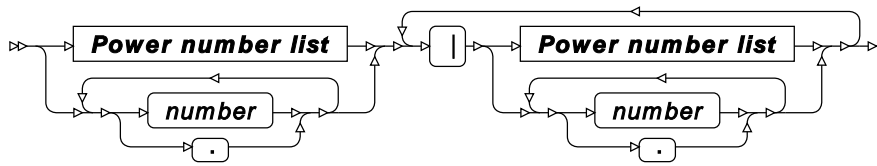
Power crossed number list



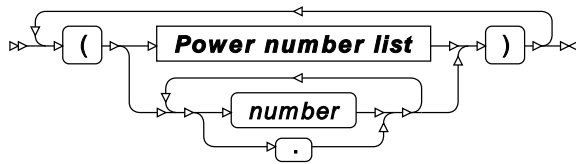
Power matched number list



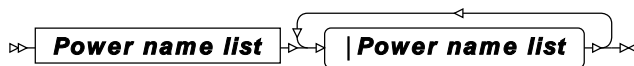
Power crossed number list



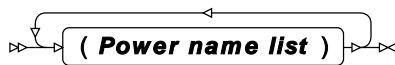
Power matched number list



Power crossed name list



Power matched name list



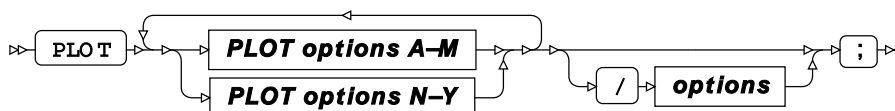
Power grouped number list



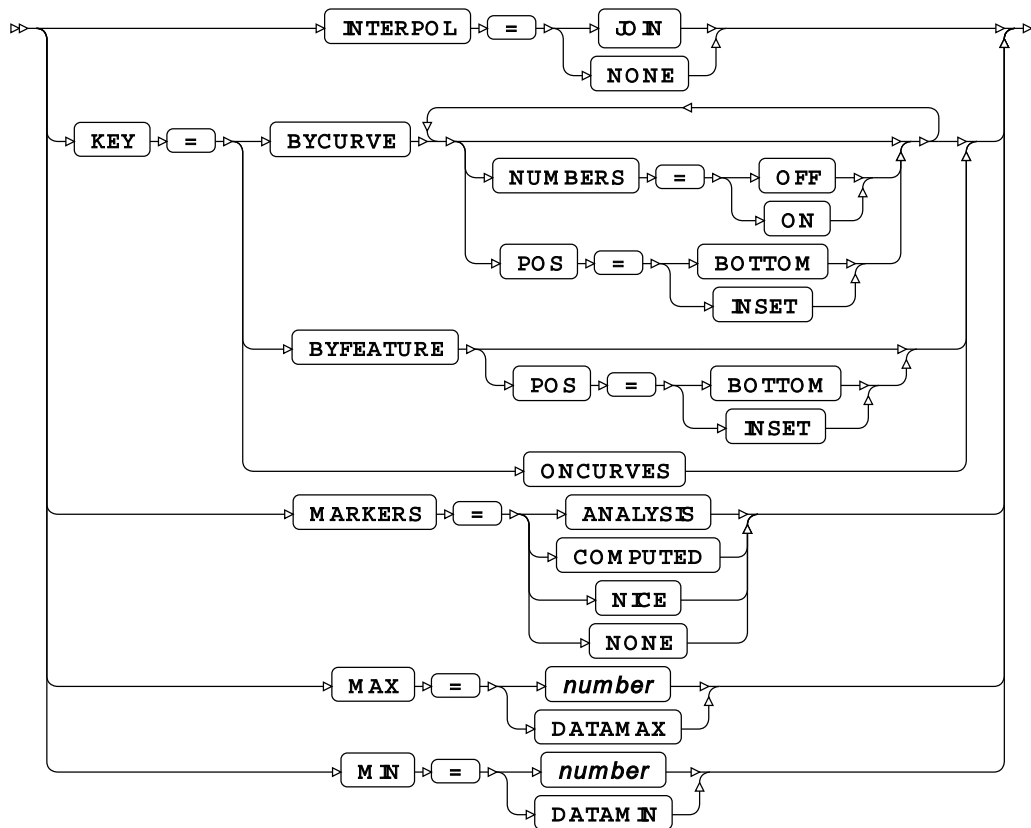
Power grouped name list



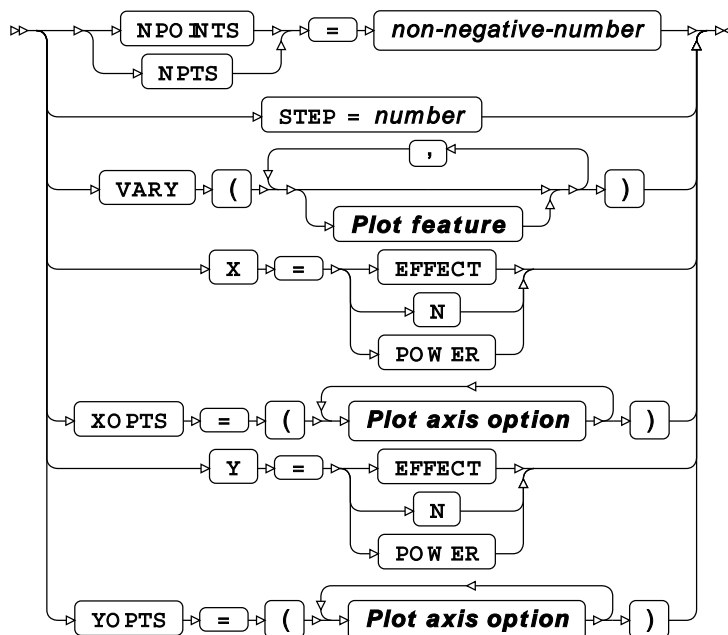
PLOT



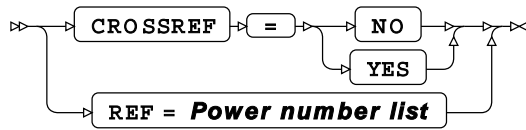
PLOT options A–M



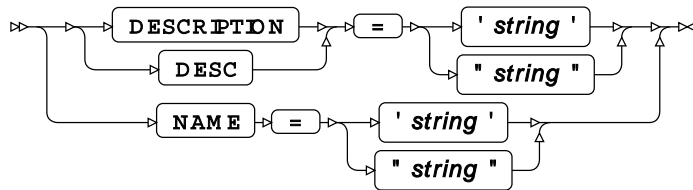
PLOT options N–Y



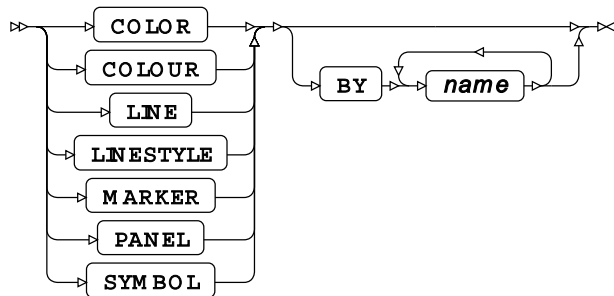
Plot axis option



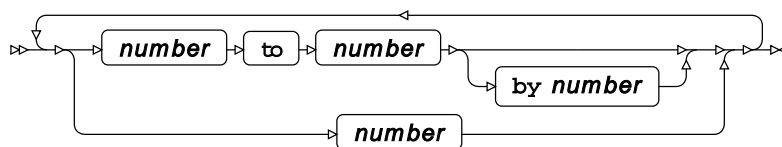
options



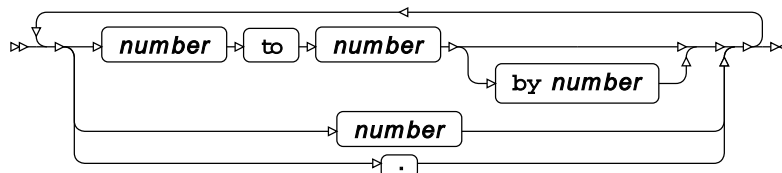
Plot feature



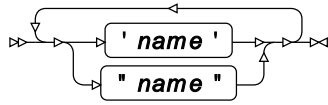
Power number list



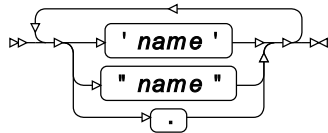
Power number list with missing



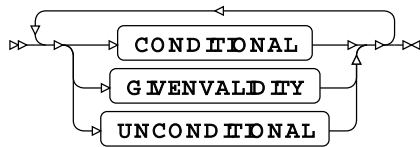
Power name list



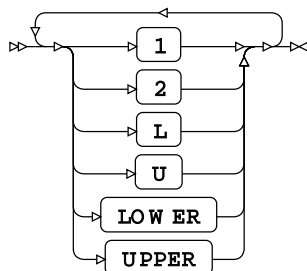
Power name list with missing



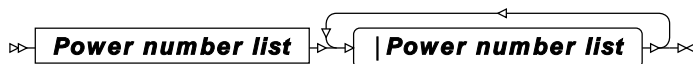
Power probtype list



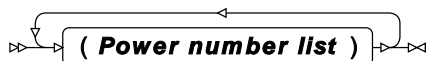
Power sides list



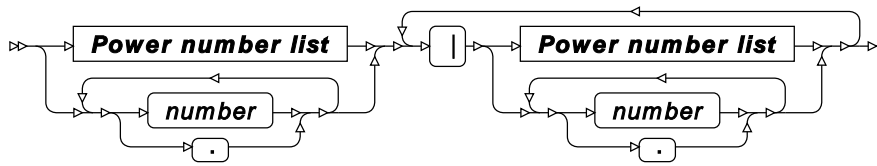
Power crossed number list



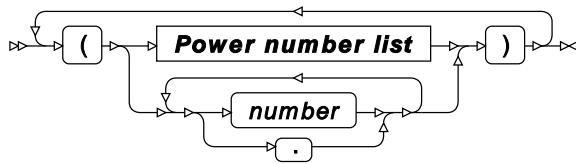
Power matched number list



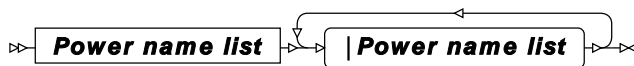
Power crossed number list



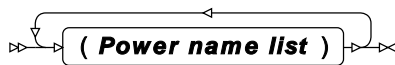
Power matched number list



Power crossed name list



Power matched name list



Power grouped number list



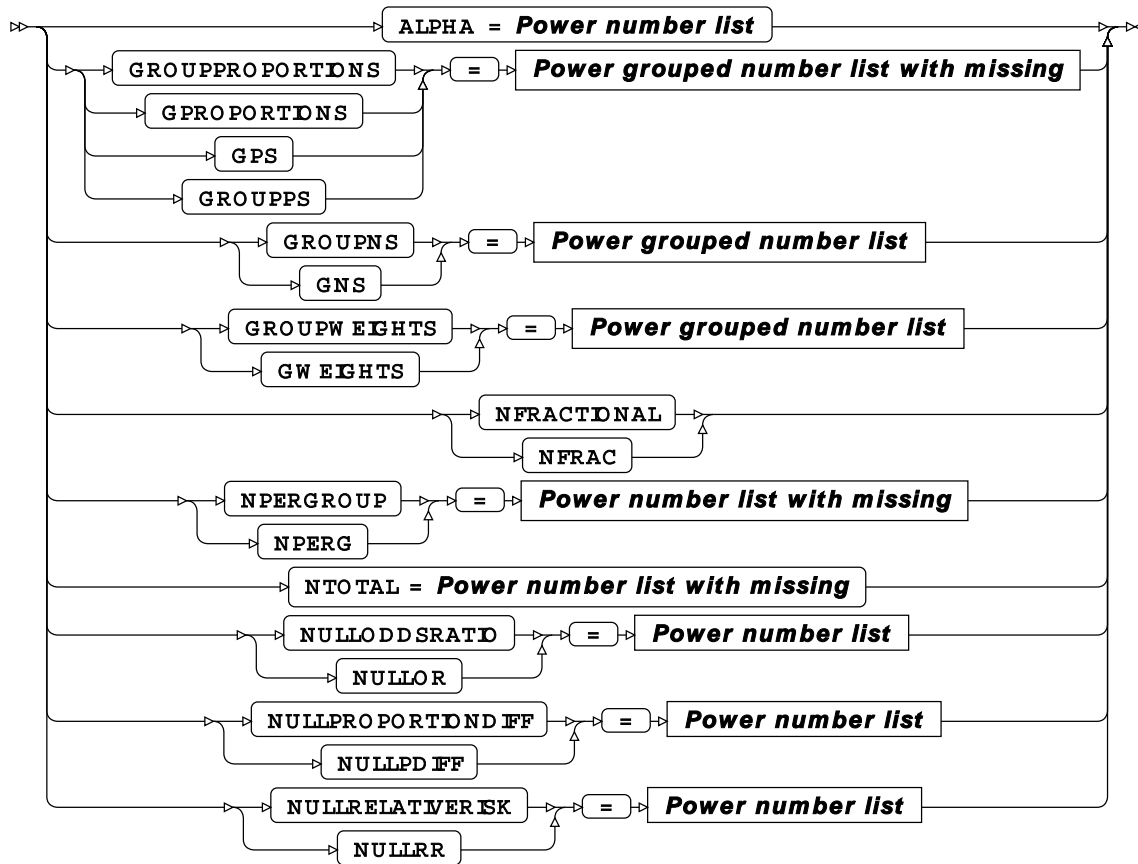
Power grouped name list



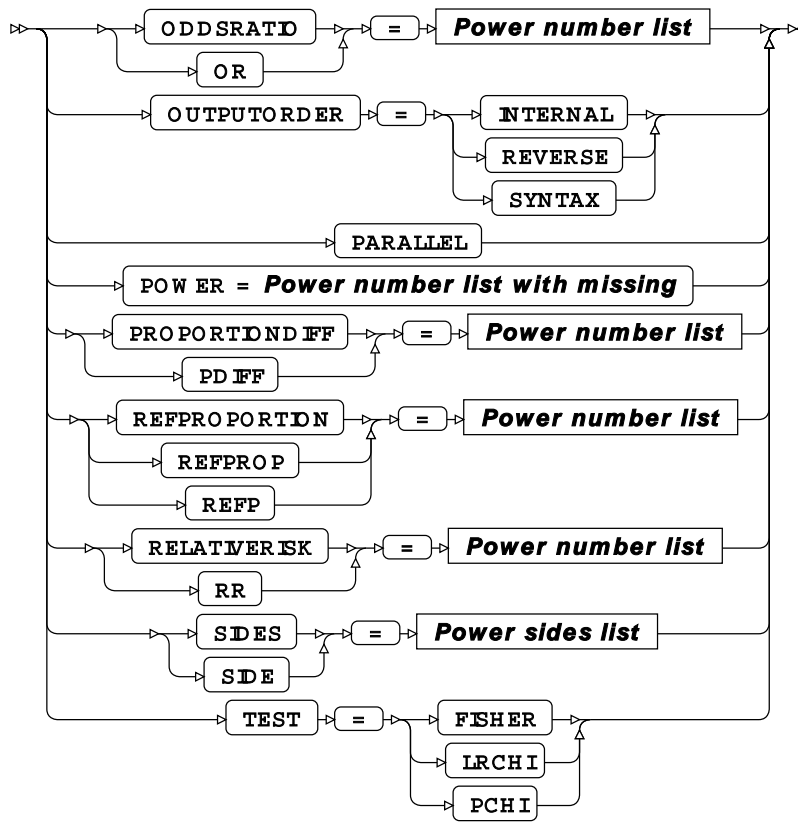
TWOSAMPLEFREQ



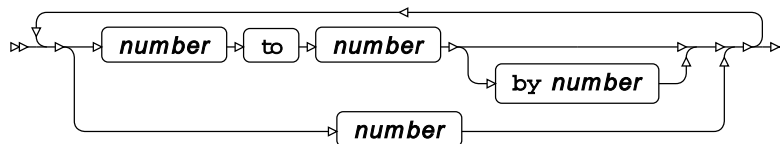
TWOSAMPLEFREQ options A–N



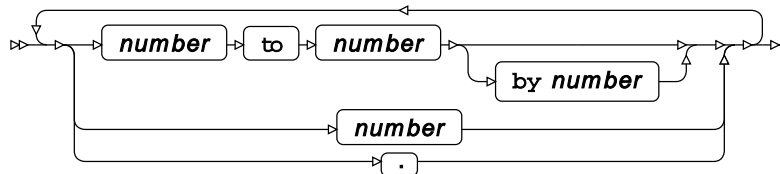
TWOSAMPLEFREQ options O–T



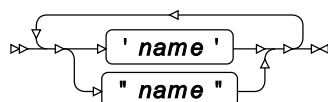
Power number list



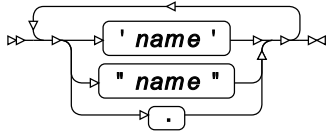
Power number list with missing



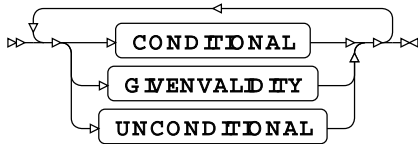
Power name list



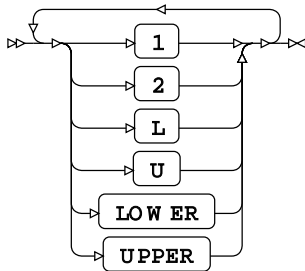
Power name list with missing



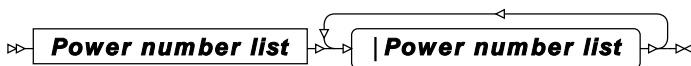
Power probtype list



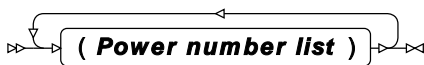
Power sides list



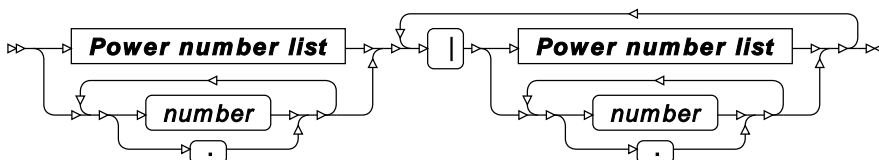
Power crossed number list



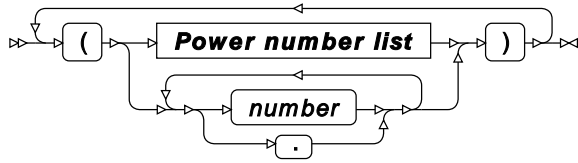
Power matched number list



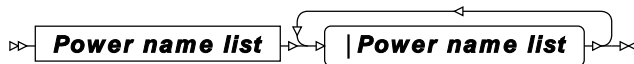
Power crossed number list with missing



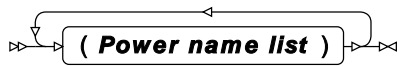
Power matched number list with missing



Power crossed name list



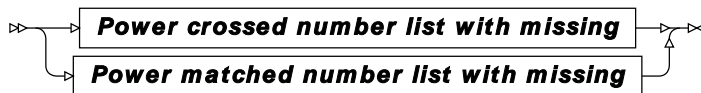
Power matched name list



Power grouped number list



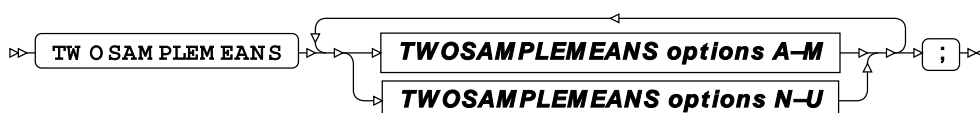
Power grouped number list with missing



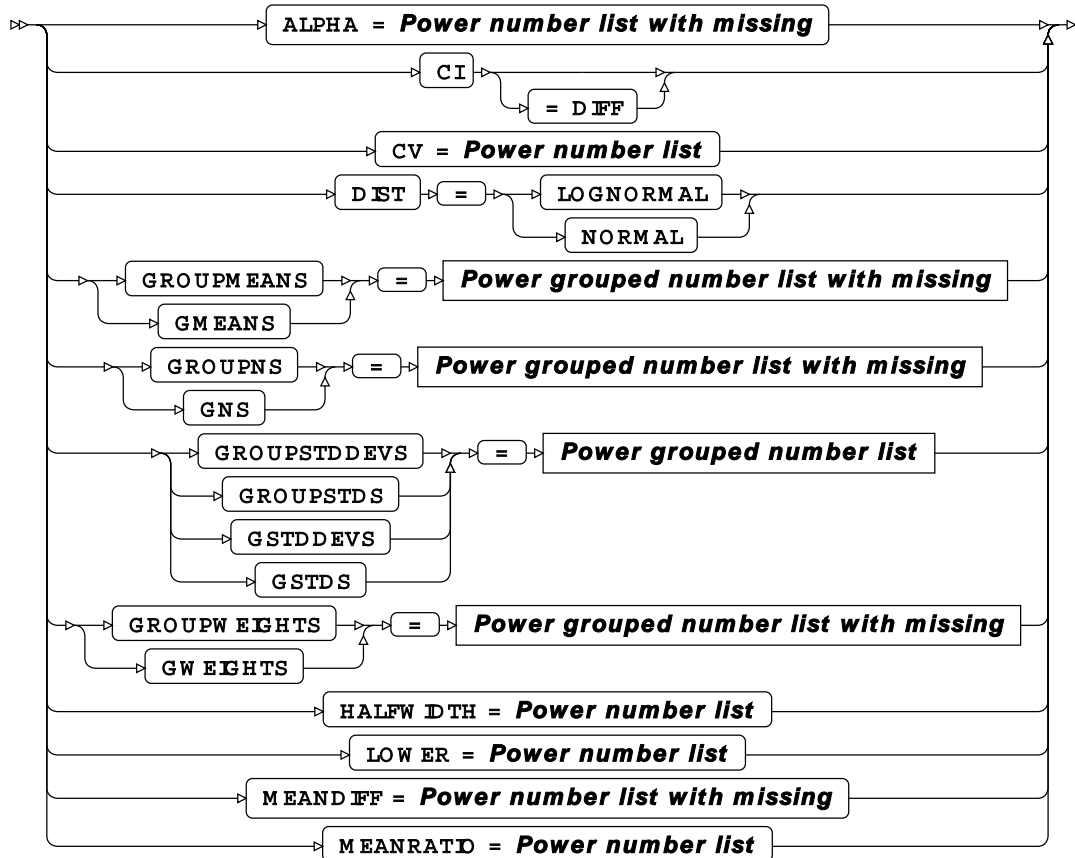
Power grouped name list



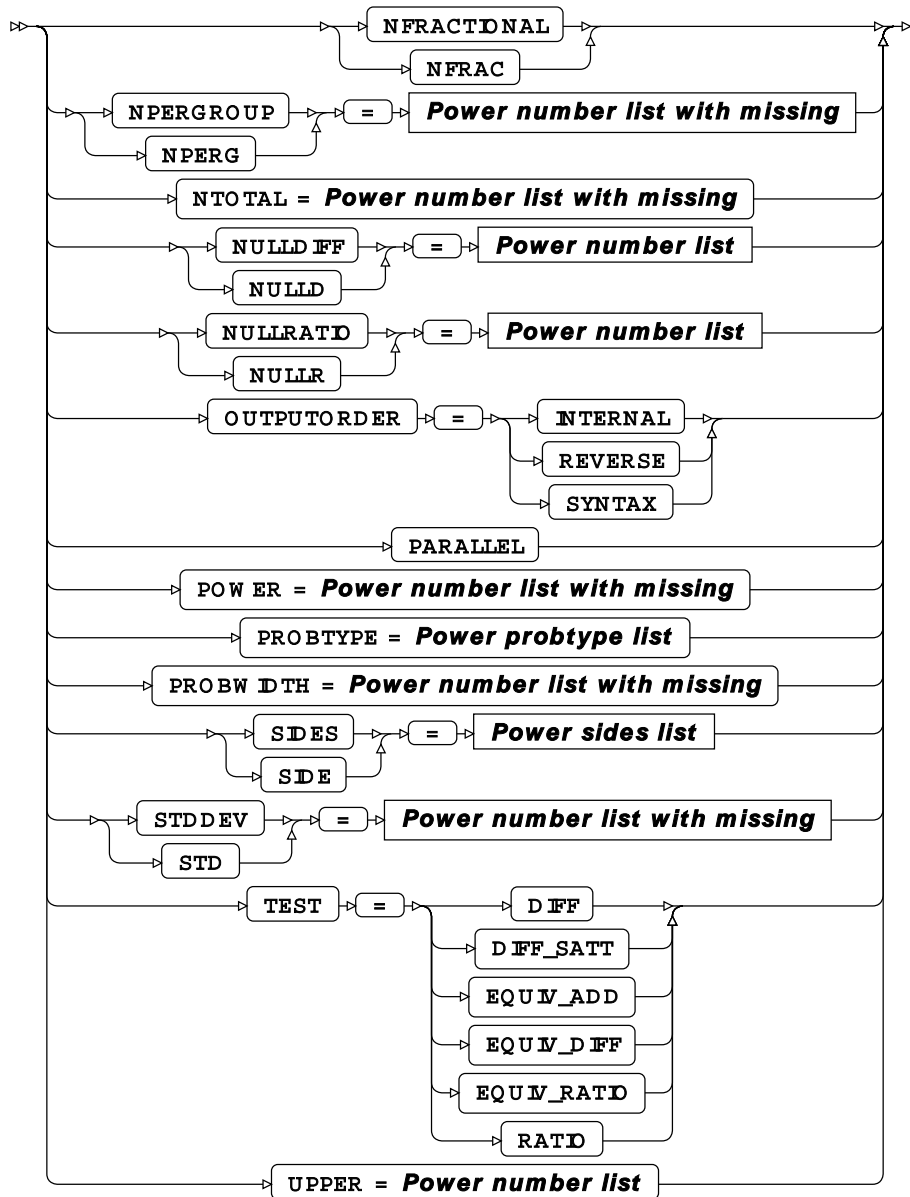
TWOSAMPLEMEANS



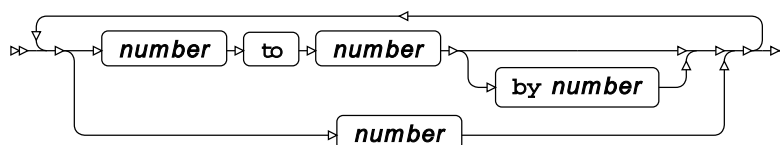
TWOSAMPLEMEANS options A–M



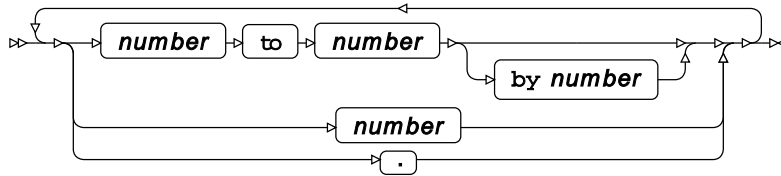
TWOSAMPLEMEANS options N–U



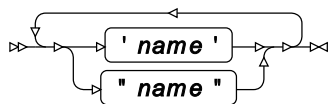
Power number list



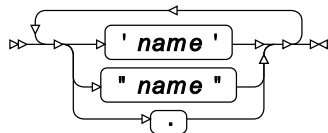
Power number list with missing



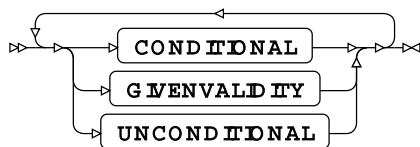
Power name list



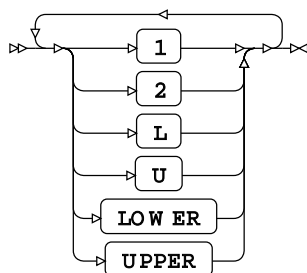
Power name list with missing



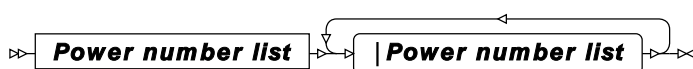
Power probtype list



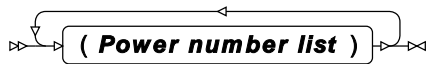
Power sides list



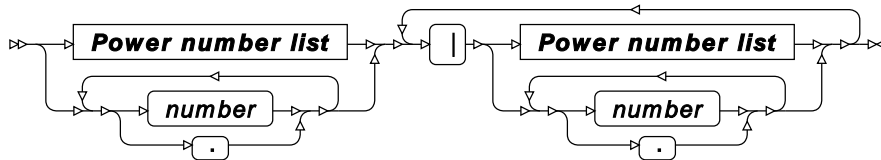
Power crossed number list



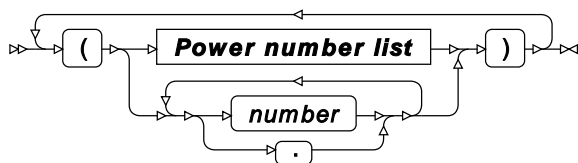
Power matched number list



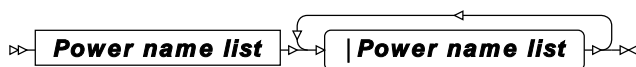
Power crossed number list with missing



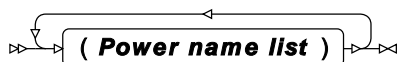
Power matched number list with missing



Power crossed name list



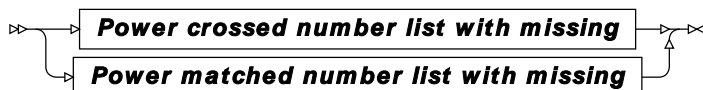
Power matched name list



Power grouped number list



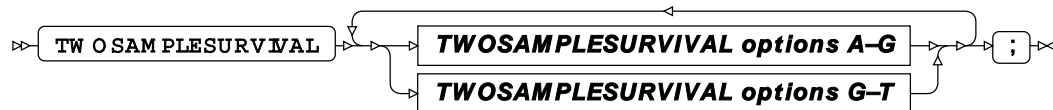
Power grouped number list with missing



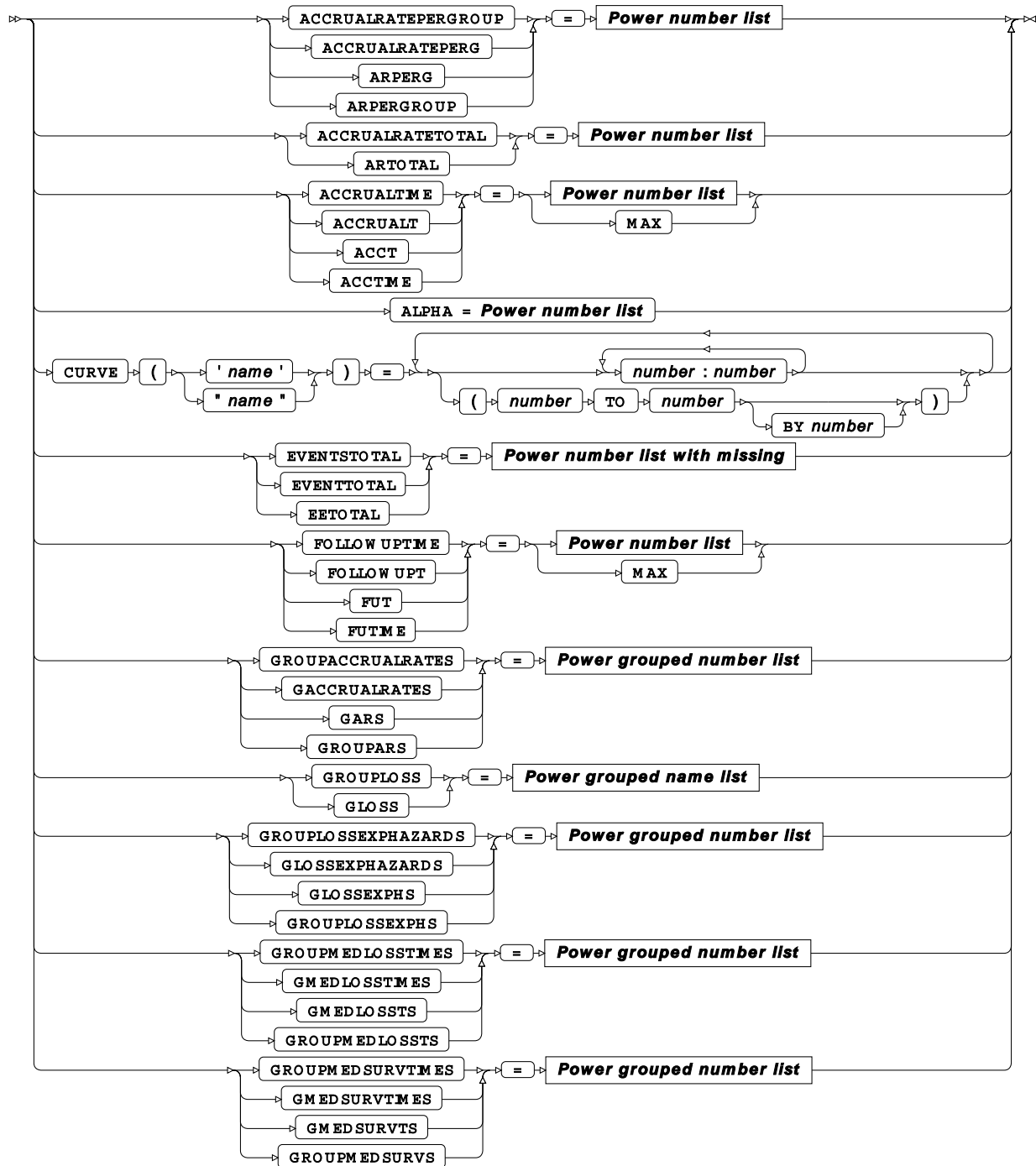
Power grouped name list



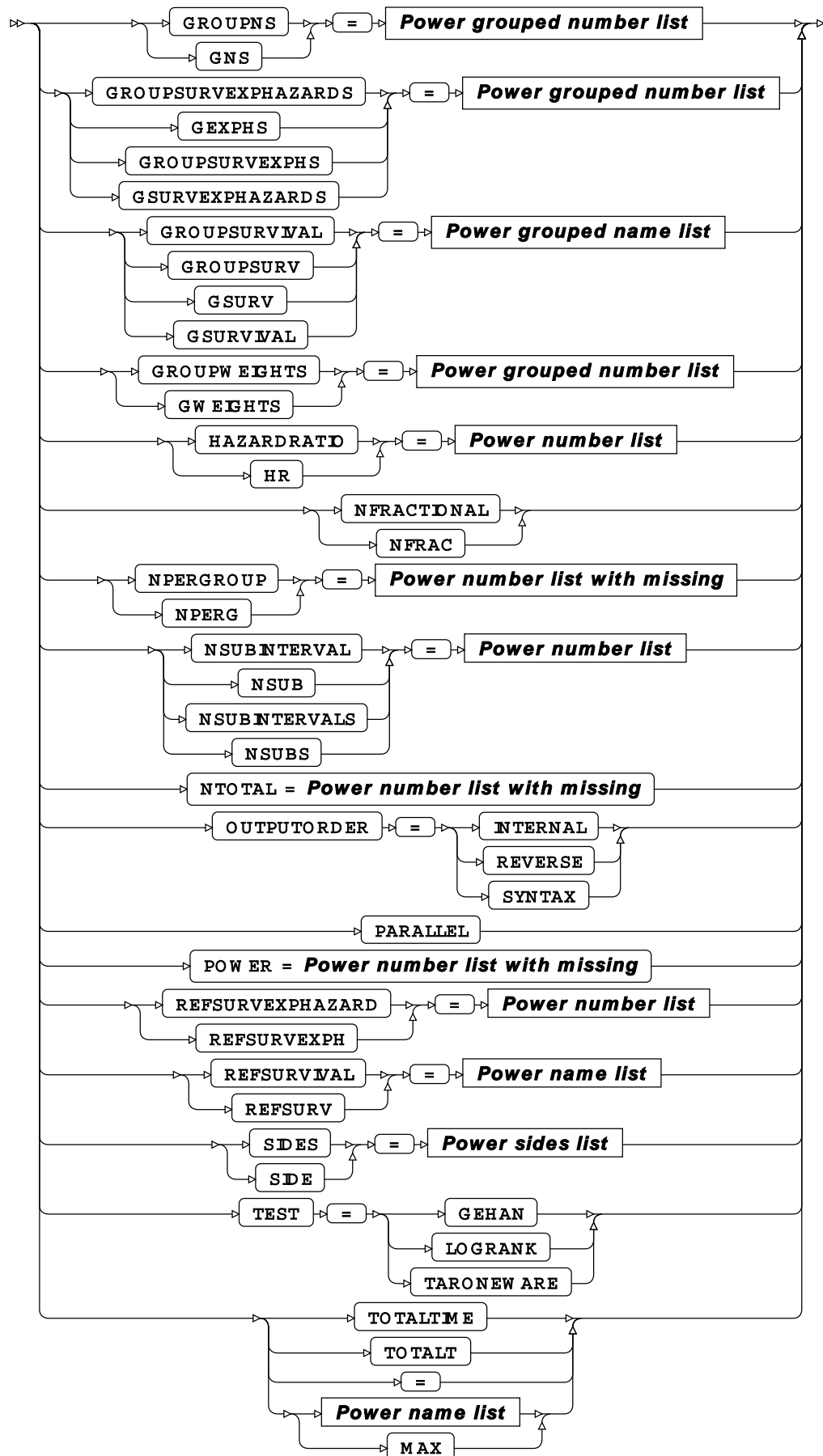
TWOSAMPLESURVIVAL



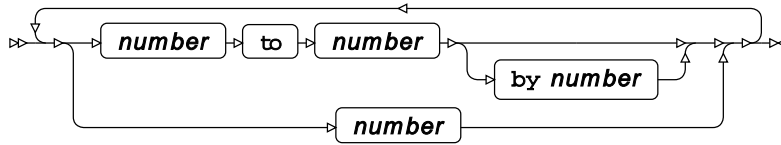
TWOSAMPLESURVIVAL options A–G



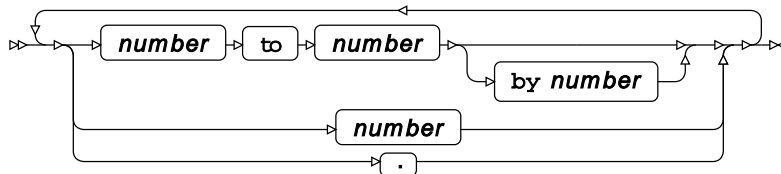
TWOSAMPLESURVIVAL options G–T



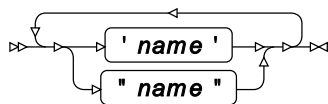
Power number list



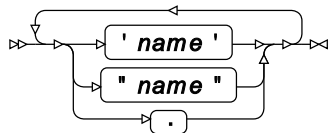
Power number list with missing



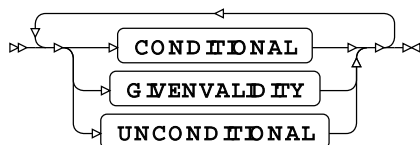
Power name list



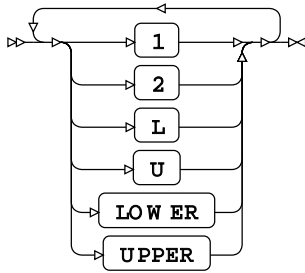
Power name list with missing



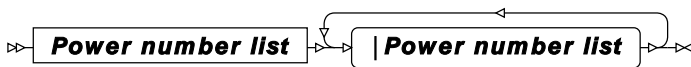
Power probtype list



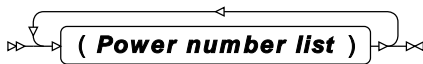
Power sides list



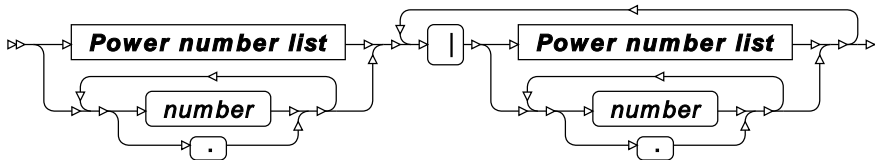
Power crossed number list



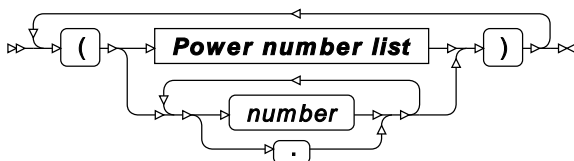
Power matched number list



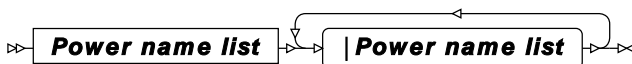
Power crossed number list



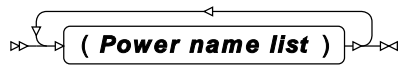
Power matched number list



Power crossed name list



Power matched name list



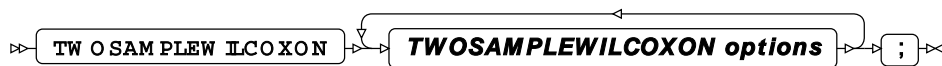
Power grouped number list



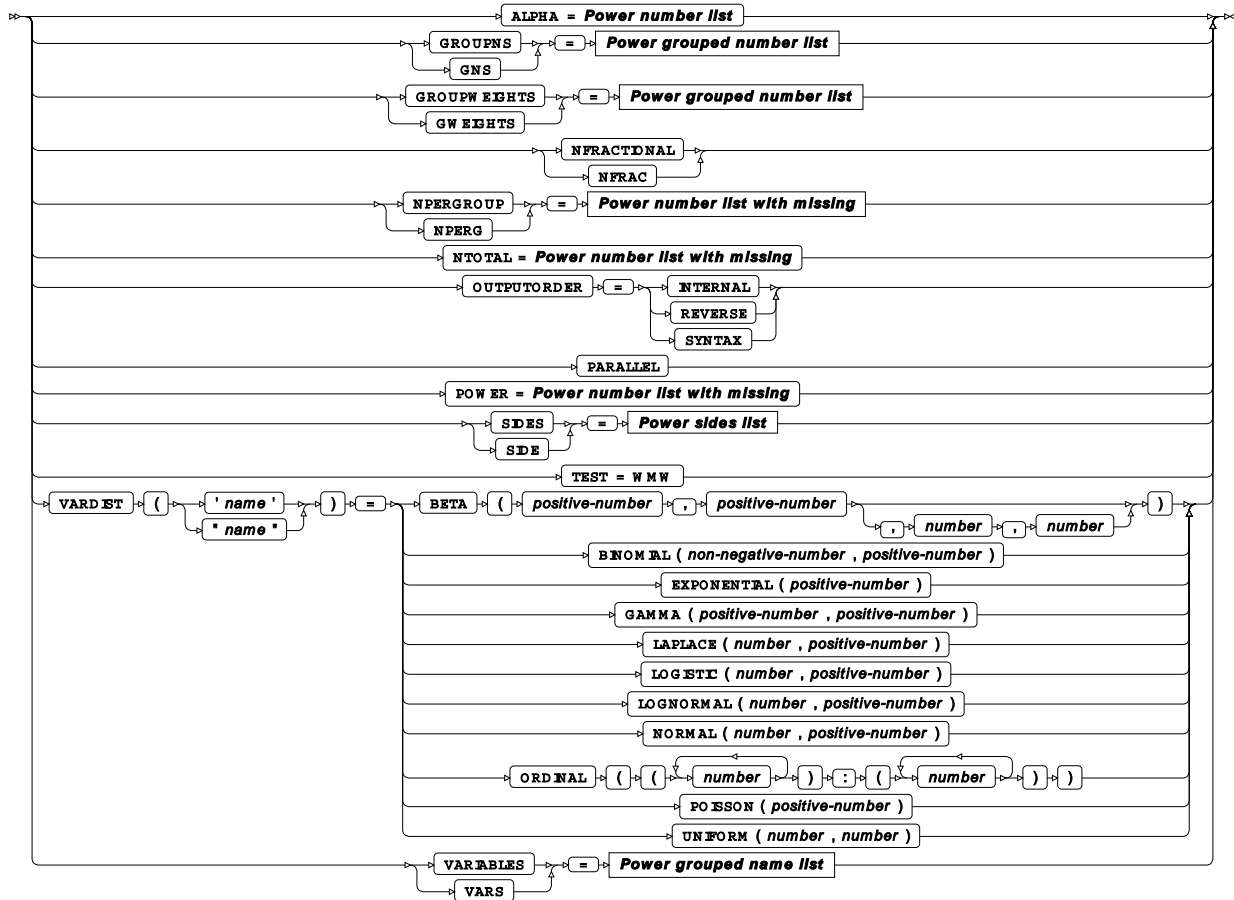
Power grouped name list



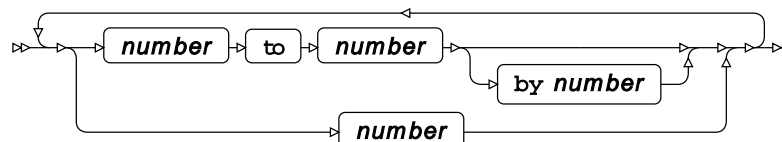
TWOSAMPLEWILCOXON



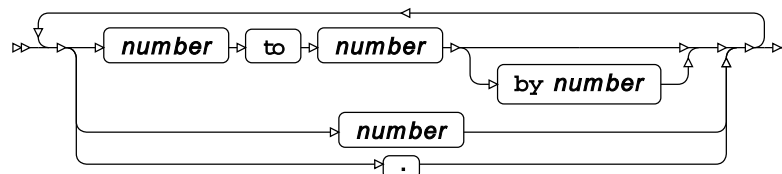
TWOSAMPLEWILCOXON options



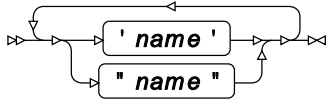
Power number list



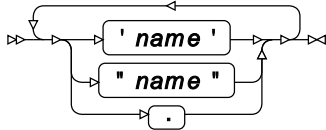
Power number list with missing



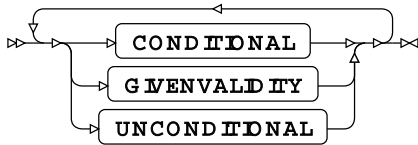
Power name list



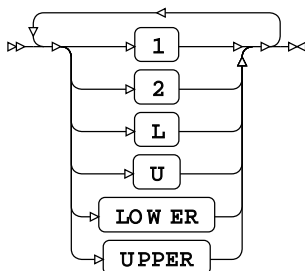
Power name list with missing



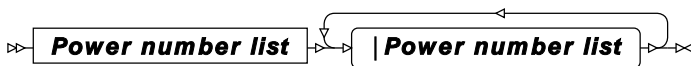
Power probtype list



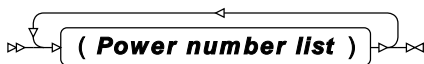
Power sides list



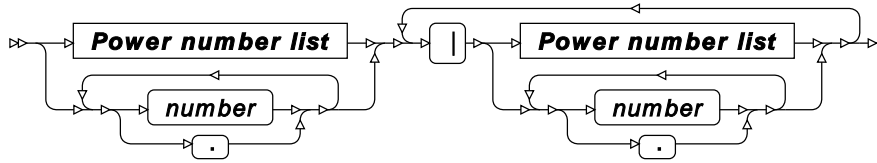
Power crossed number list



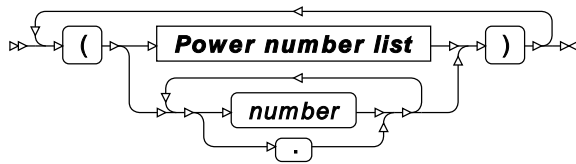
Power matched number list



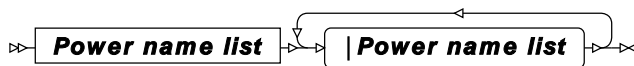
Power crossed number list



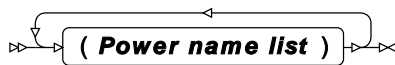
Power matched number list



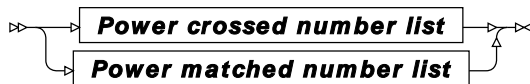
Power crossed name list



Power matched name list



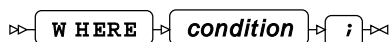
Power grouped number list



Power grouped name list



WHERE

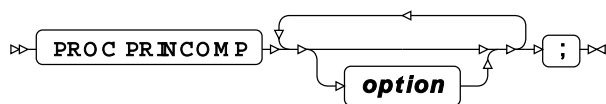


PRINCOMP procedure

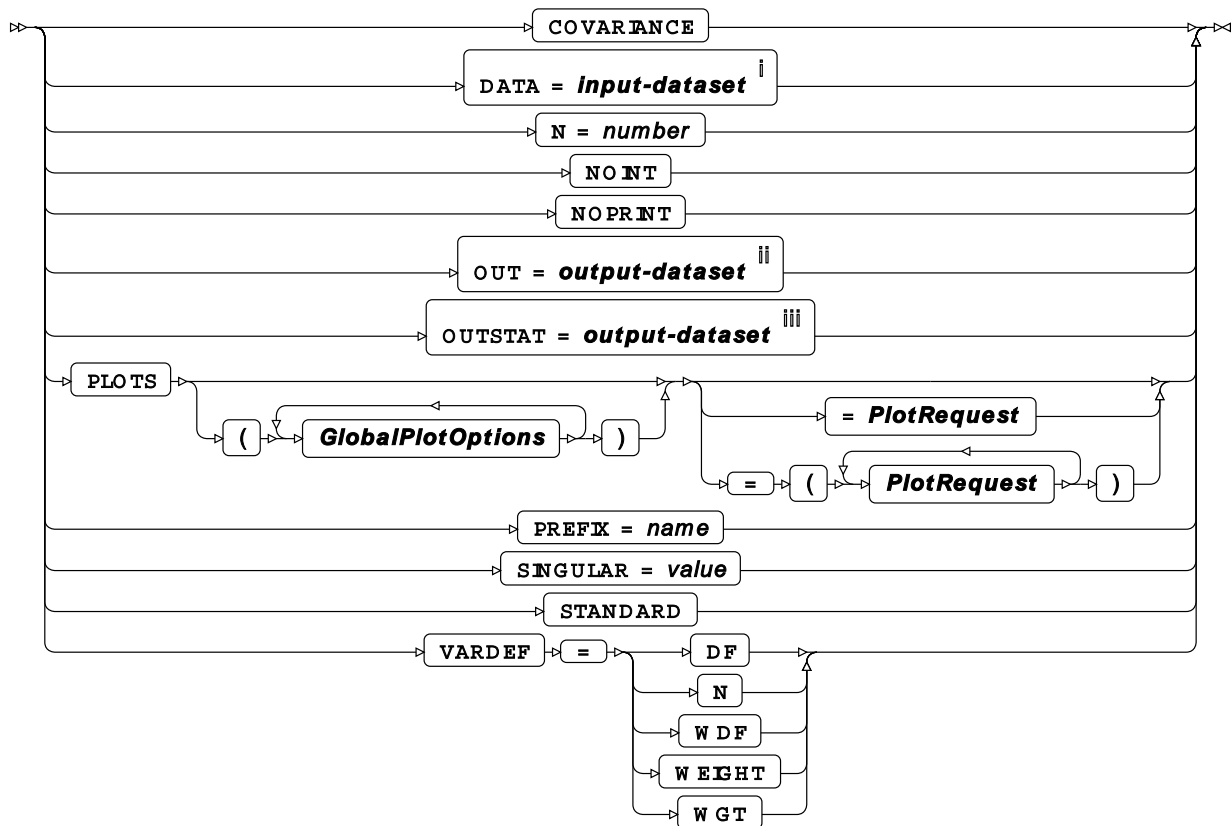
Supported statements

- *PROC PRINCOMP* [↗](#) (page 3072)
- *ATTRIB* [↗](#) (page 3074)
- *BY* [↗](#) (page 3075)
- *FORMAT* [↗](#) (page 3075)
- *FREQ* [↗](#) (page 3075)
- *ID* [↗](#) (page 3075)
- *INFORMAT* [↗](#) (page 3075)
- *LABEL* [↗](#) (page 3076)
- *PARTIAL* [↗](#) (page 3076)
- *VAR* [↗](#) (page 3076)
- *WEIGHT* [↗](#) (page 3076)
- *WHERE* [↗](#) (page 3076)

PROC PRINCOMP



option

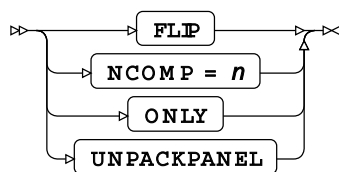


ⁱ See *Input dataset* [↗](#) (page 16).

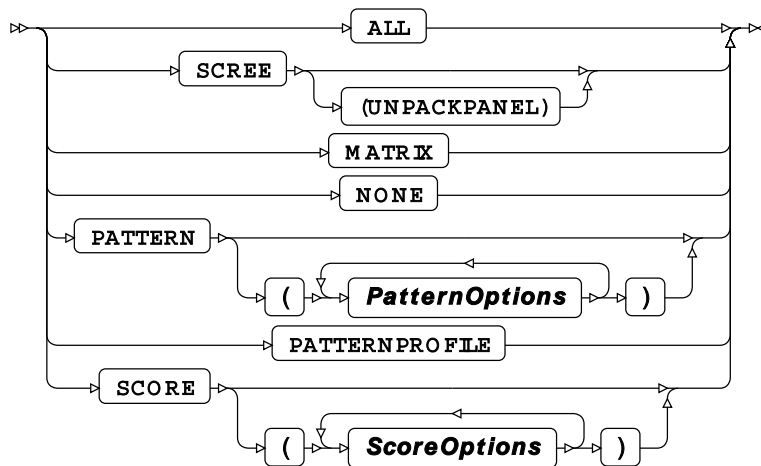
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

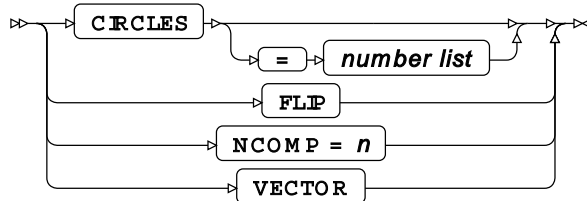
GlobalPlotOptions



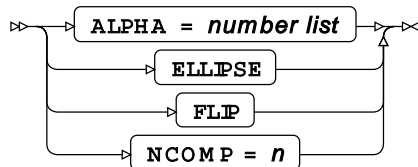
PlotRequest



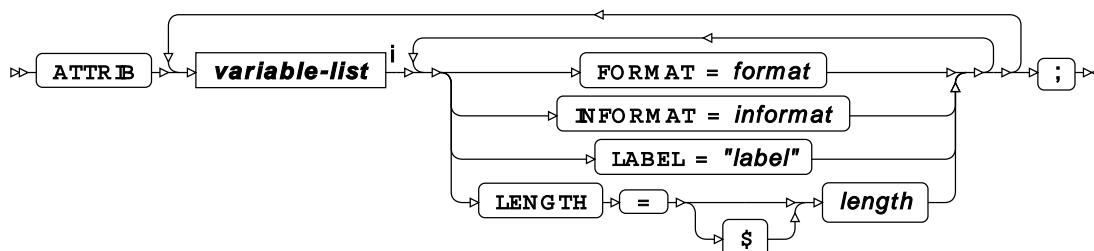
PatternOptions



ScoreOptions

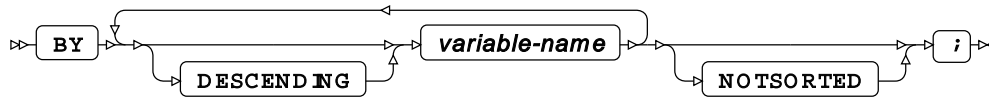


ATTRIB

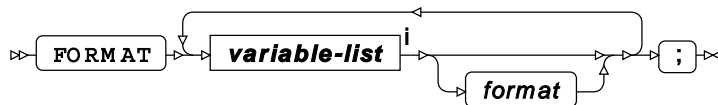


ⁱ See [Variable Lists](#) (page 32).

BY

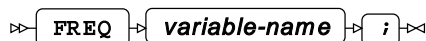


FORMAT



ⁱ See [Variable Lists](#) (page 32).

FREQ



ID

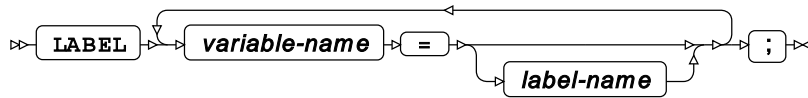


INFORMAT

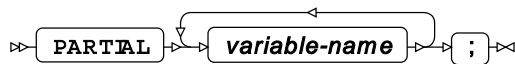


ⁱ See [Variable Lists](#) (page 32).

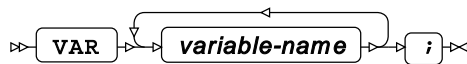
LABEL



PARTIAL



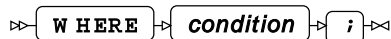
VAR



WEIGHT



WHERE



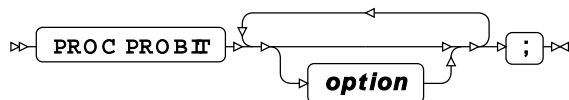
PROBIT procedure

Supported statements

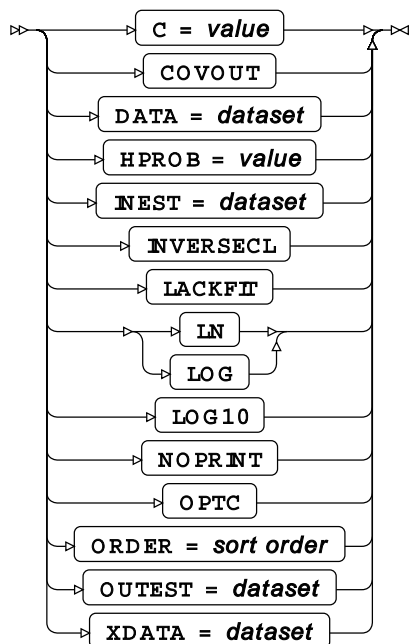
- *PROC PROBIT* [↗](#) (page 3077)
- *ATTRIB* [↗](#) (page 3078)
- *BY* [↗](#) (page 3078)
- *CLASS* [↗](#) (page 3078)
- *CODE* [↗](#) (page 3079)

- *ESTIMATE* [↗](#) (page 3080)
- *FORMAT* [↗](#) (page 3081)
- *INFORMAT* [↗](#) (page 3082)
- *LABEL* [↗](#) (page 3082)
- *MODEL* [↗](#) (page 3082)
- *OUTPUT* [↗](#) (page 3084)
- *WEIGHT* [↗](#) (page 3085)
- *WHERE* [↗](#) (page 3085)

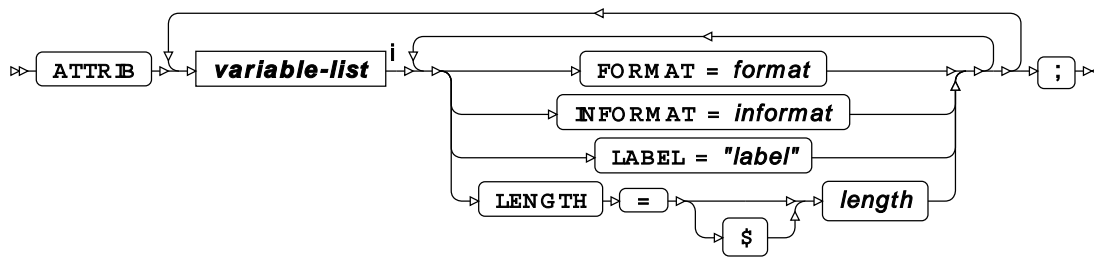
PROC PROBIT



option

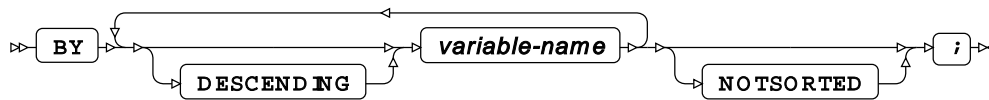


ATTRIB

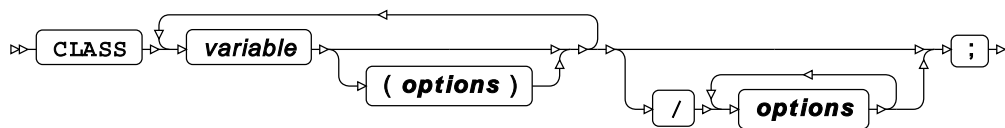


ⁱ See *Variable Lists* [↗](#) (page 32).

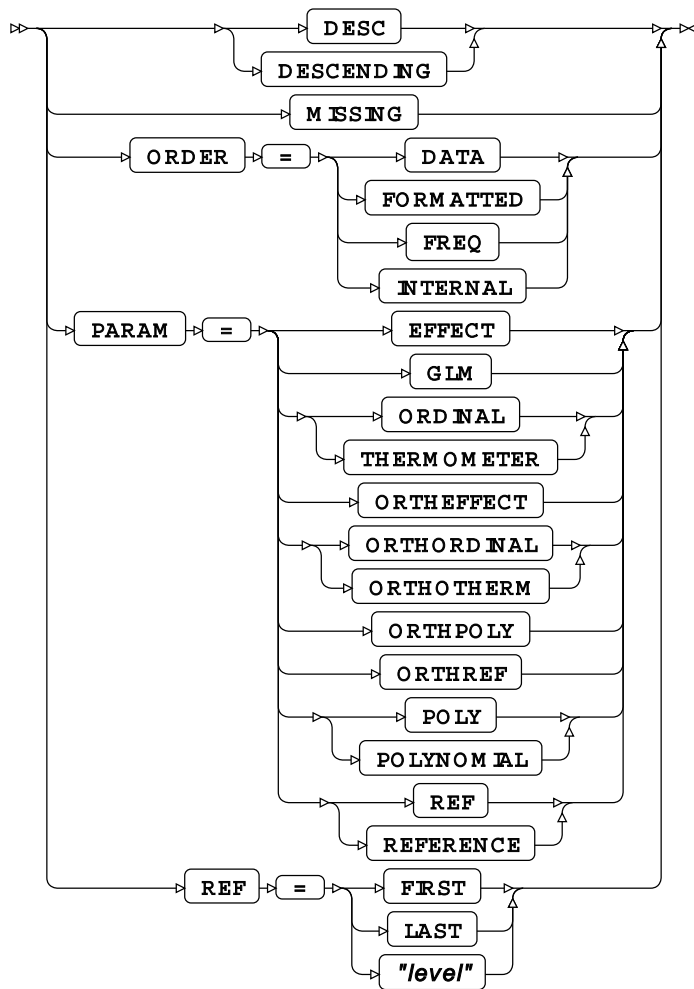
BY



CLASS



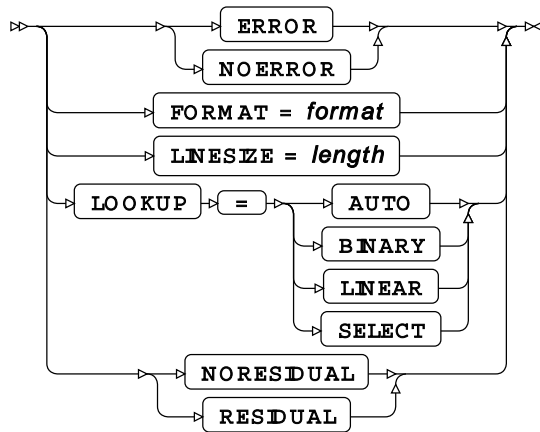
options



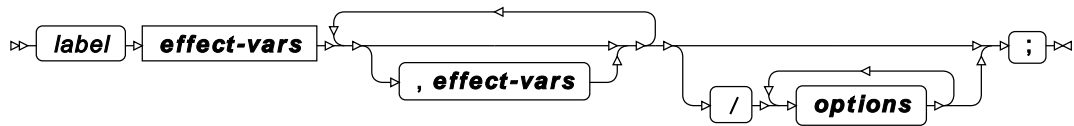
CODE



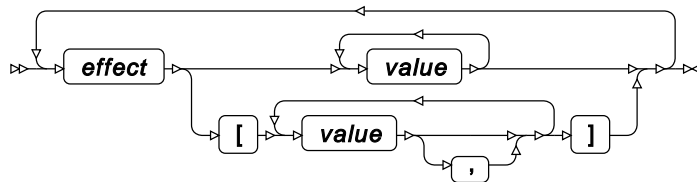
options



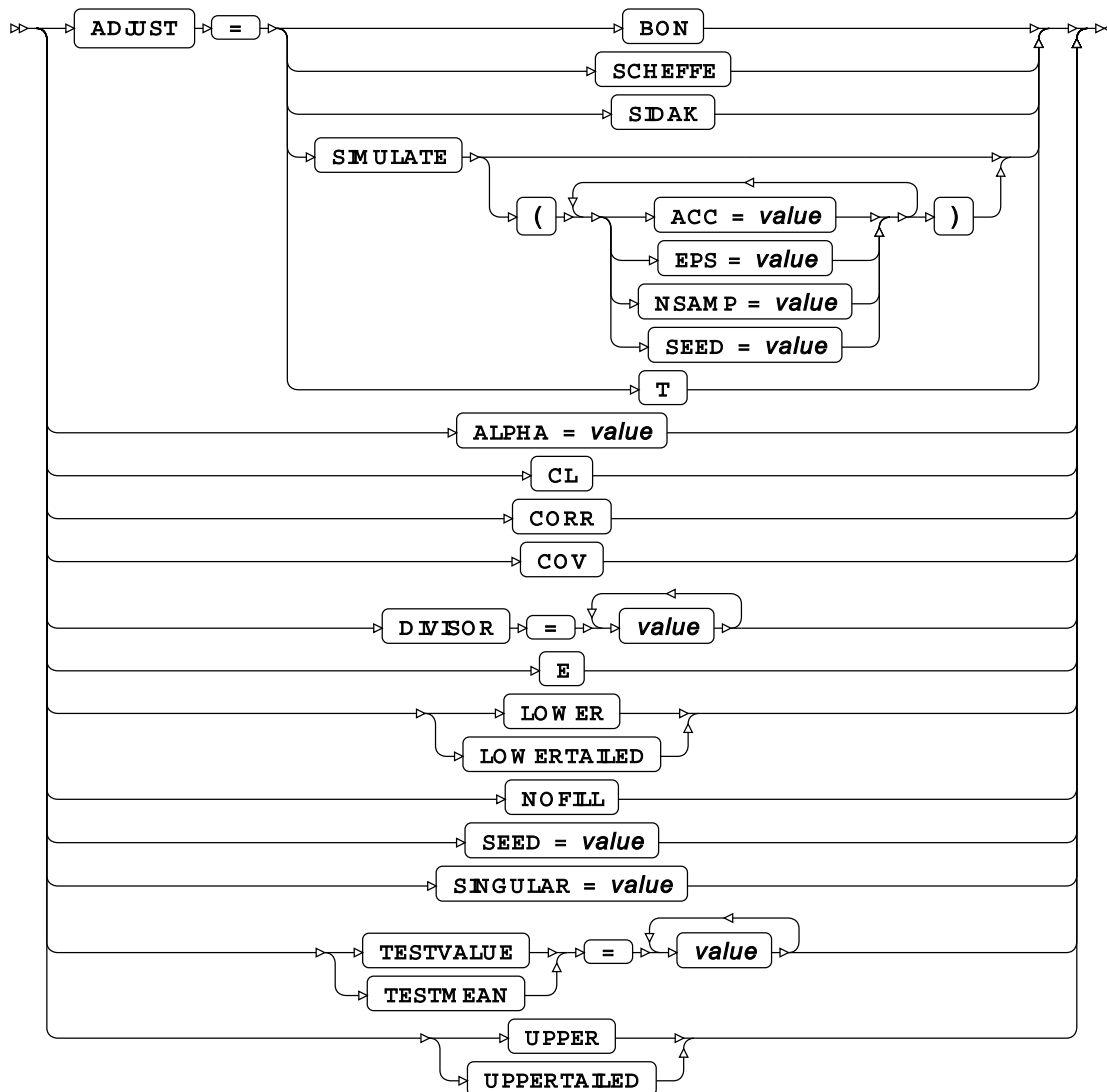
ESTIMATE



effect-vars



options



FORMAT



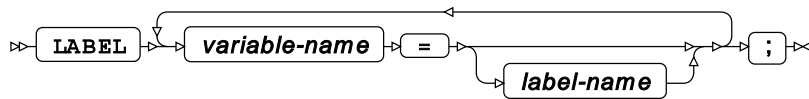
ⁱ See [Variable Lists](#) (page 32).

INFORMAT

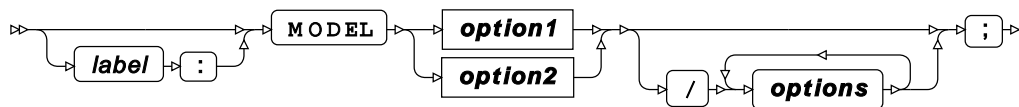


ⁱ See *Variable Lists* [↗](#) (page 32).

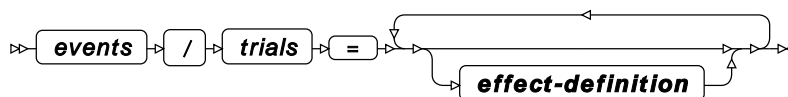
LABEL



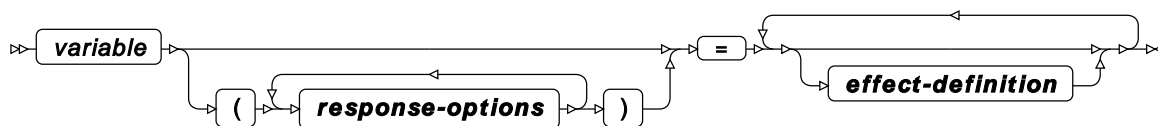
MODEL



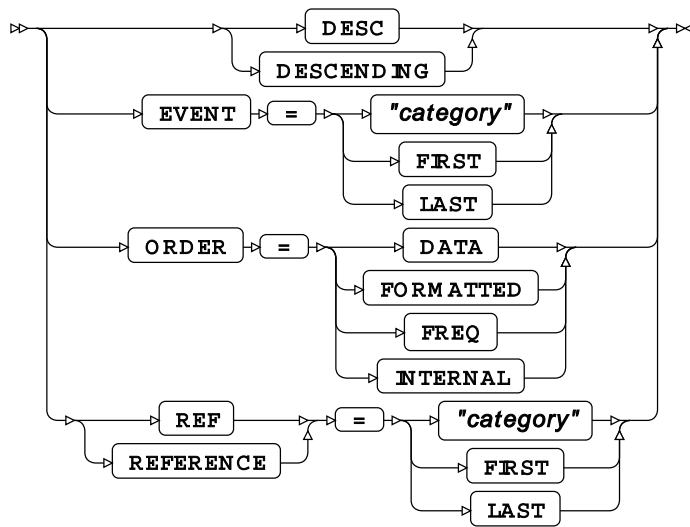
option1



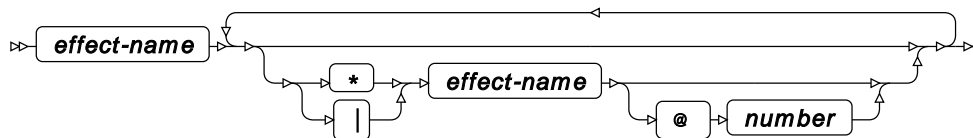
option2



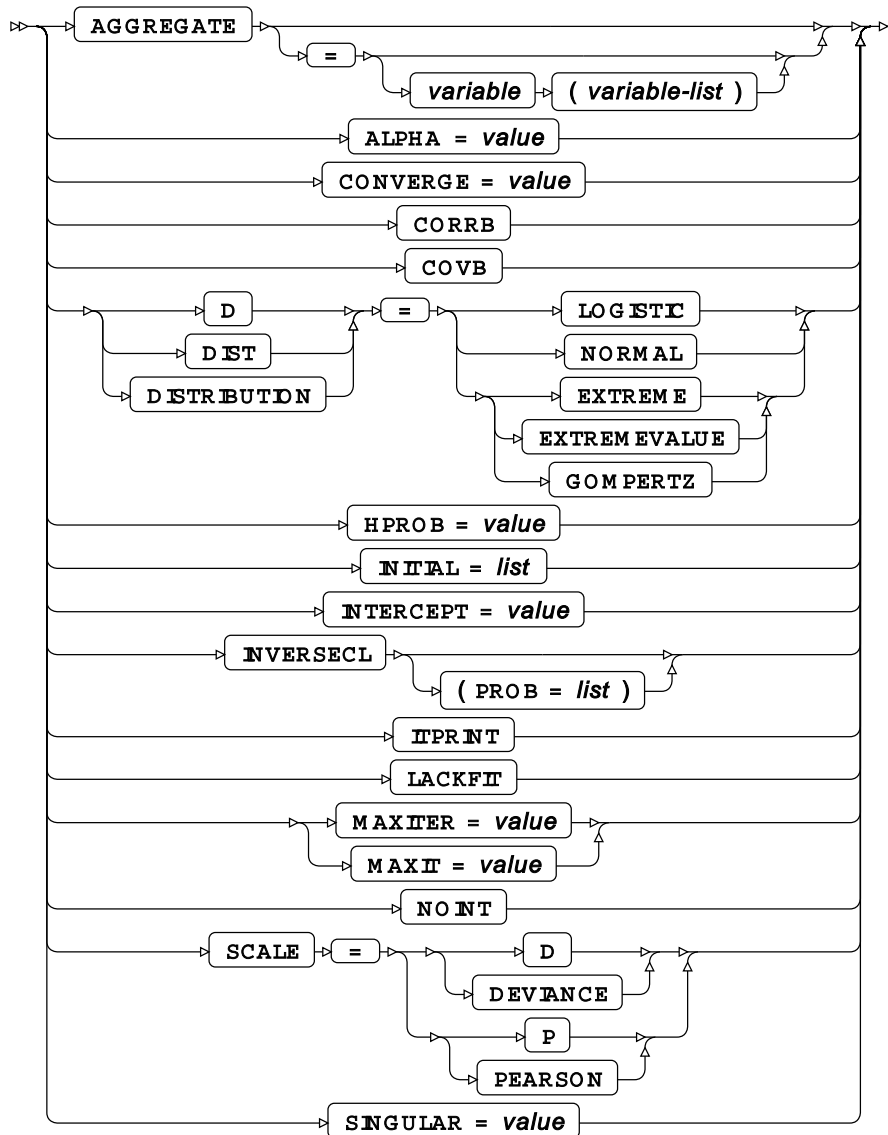
response-options



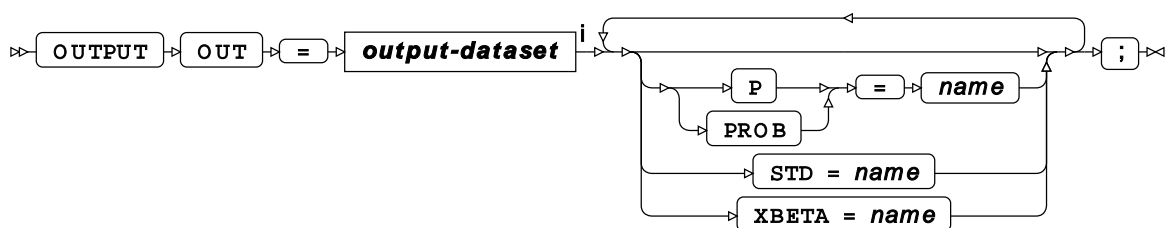
effect-definition



options



OUTPUT



ⁱ See [Output dataset](#) (page 16).

WEIGHT



WHERE

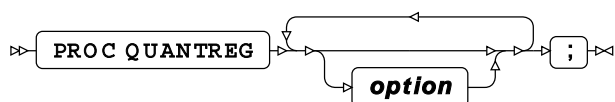


QUANTREG procedure

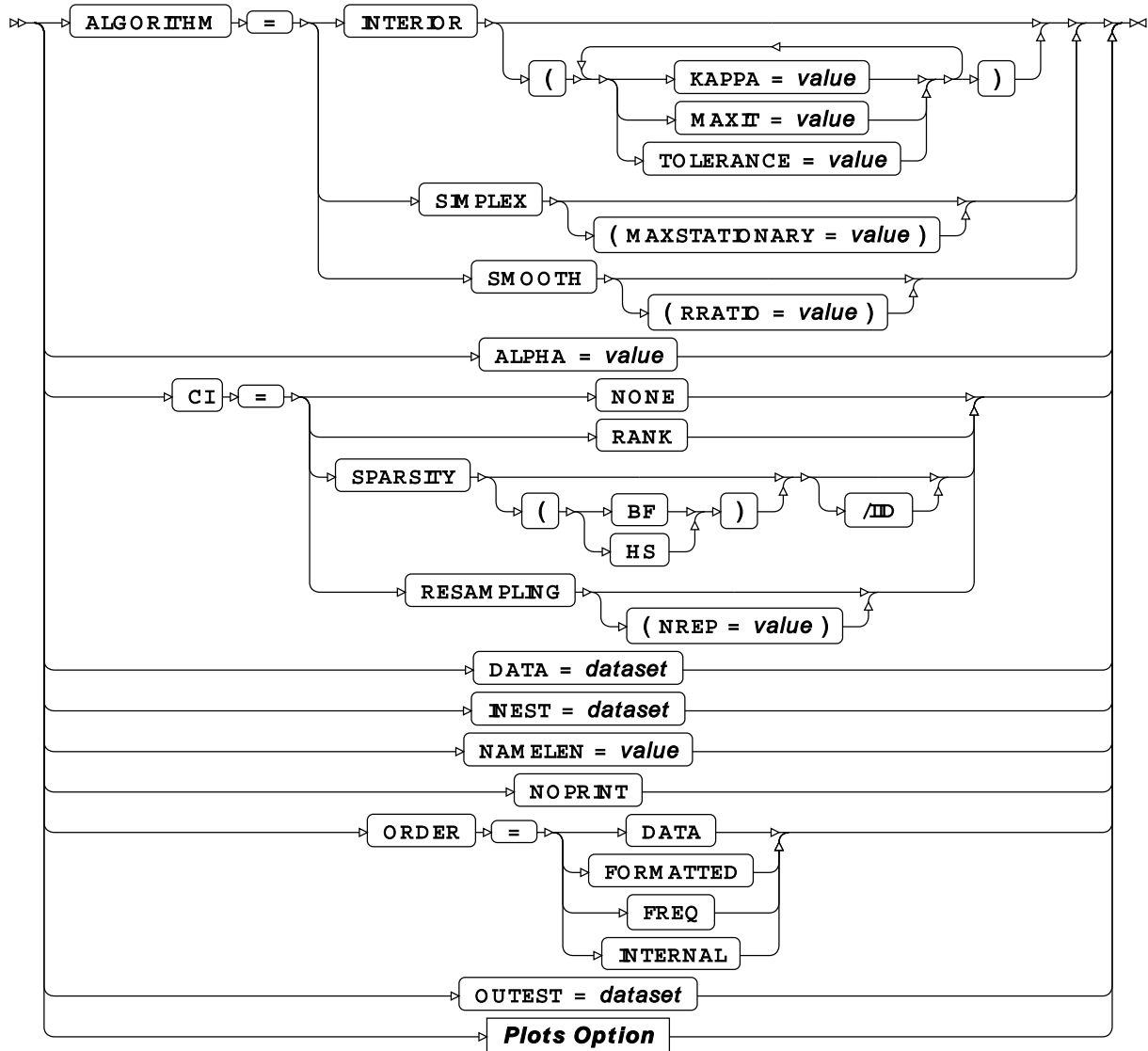
Supported statements

- *PROC QUANTREG* [↗](#) (page 3085)
- *ATTRIB* [↗](#) (page 3087)
- *BY* [↗](#) (page 3087)
- *CLASS* [↗](#) (page 3087)
- *ESTIMATE* [↗](#) (page 3088)
- *FORMAT* [↗](#) (page 3089)
- *INFORMAT* [↗](#) (page 3090)
- *LABEL* [↗](#) (page 3090)
- *MODEL* [↗](#) (page 3090)
- *OUTPUT* [↗](#) (page 3092)
- *TEST* [↗](#) (page 3092)
- *WEIGHT* [↗](#) (page 3093)
- *WHERE* [↗](#) (page 3093)

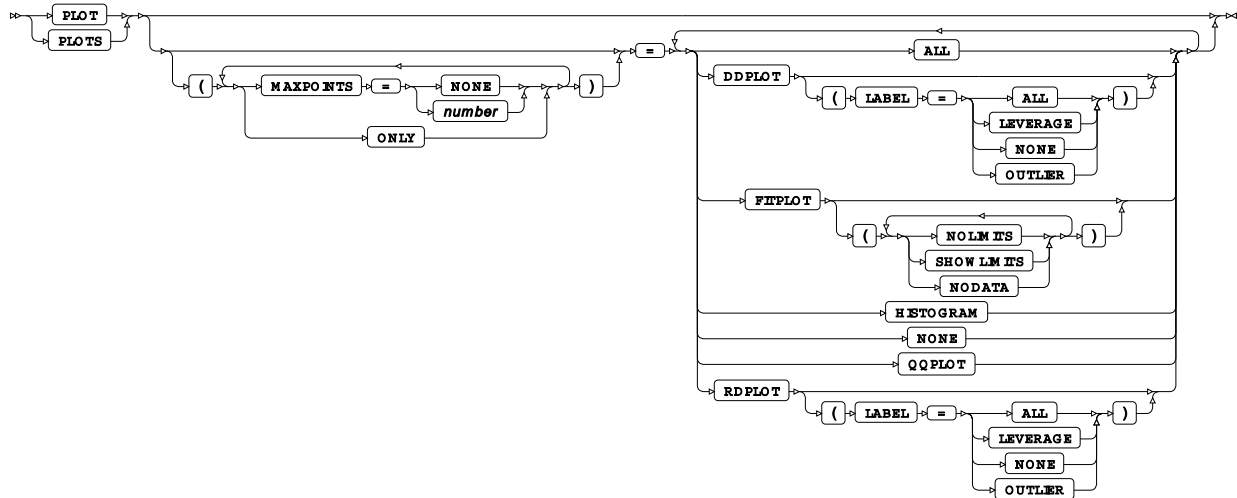
PROC QUANTREG



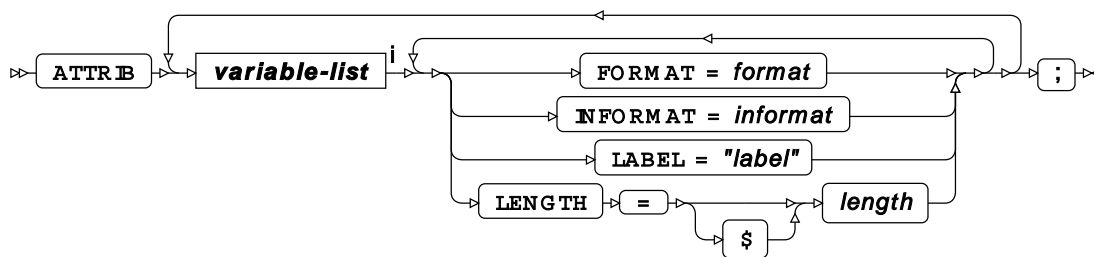
option



Plots Option

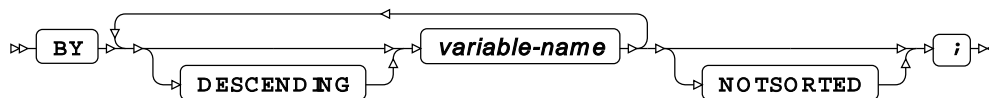


ATTRIB

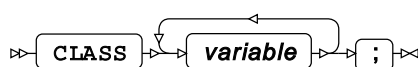


ⁱ See [Variable Lists](#) (page 32).

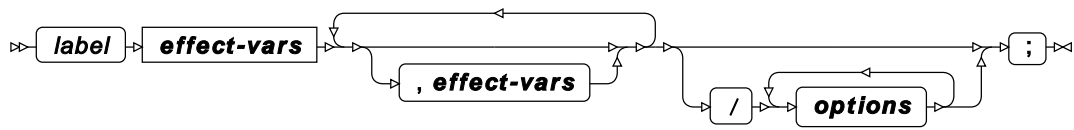
BY



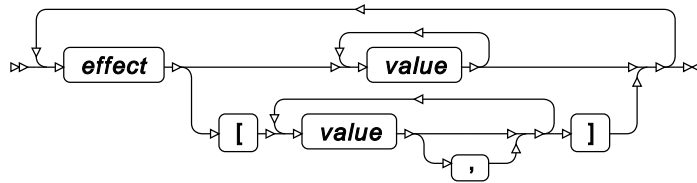
CLASS



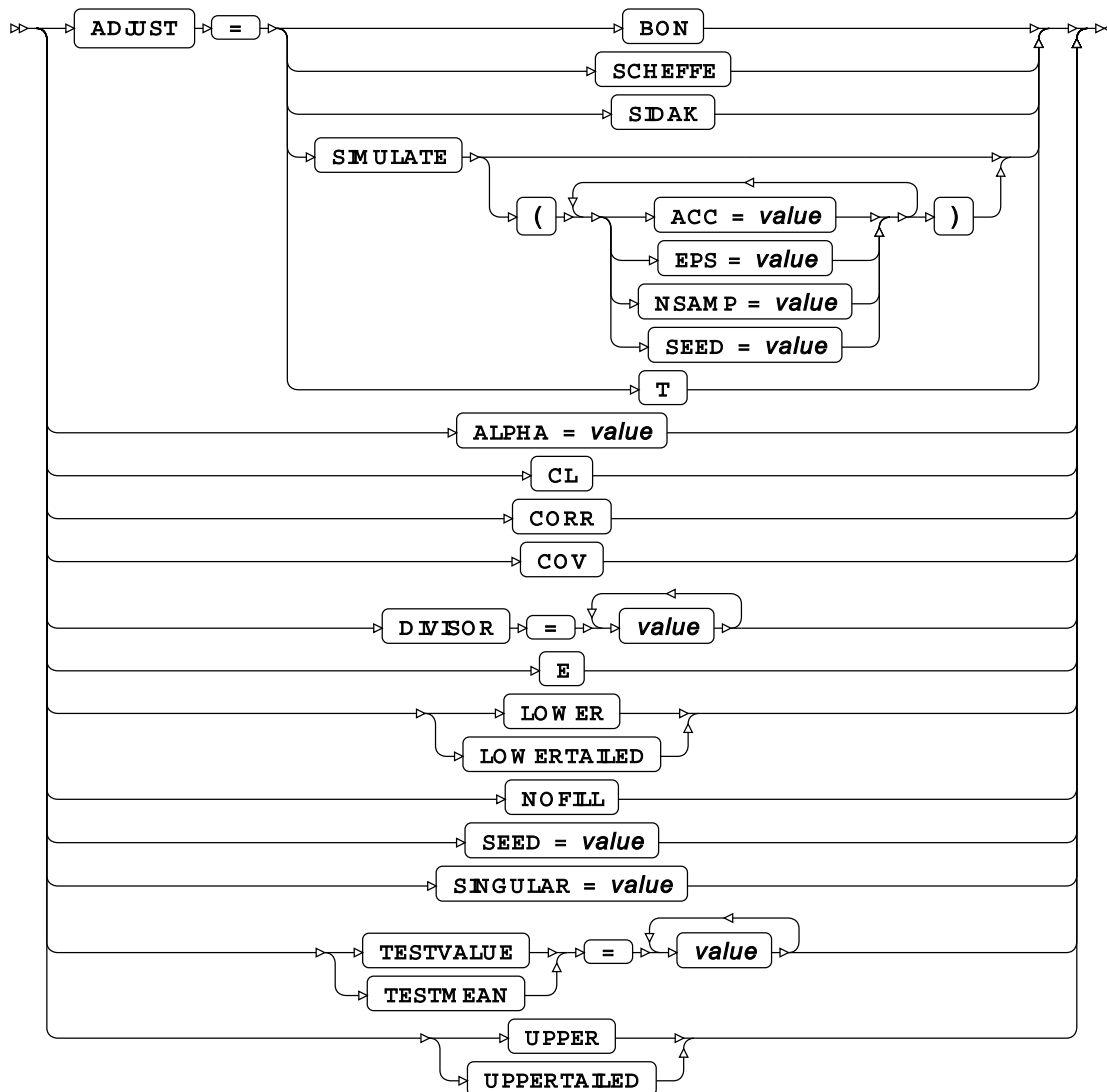
ESTIMATE



effect-vars



options

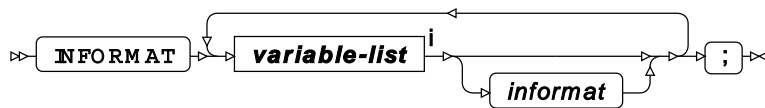


FORMAT



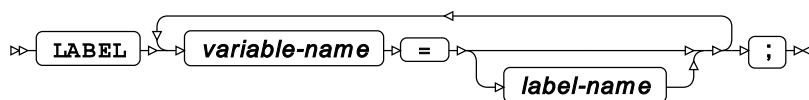
ⁱ See [Variable Lists](#) (page 32).

INFORMAT

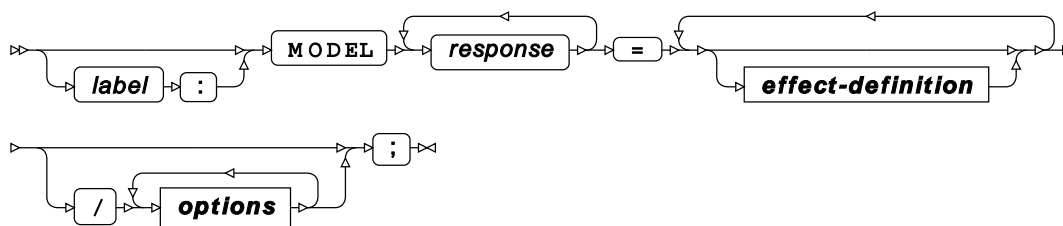


ⁱ See *Variable Lists* [↗](#) (page 32).

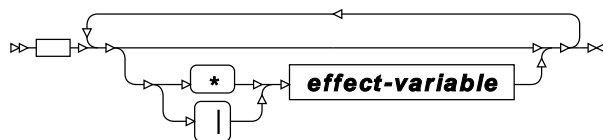
LABEL



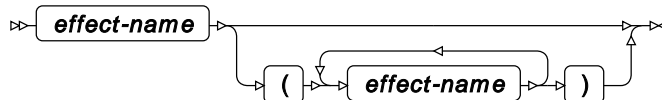
MODEL



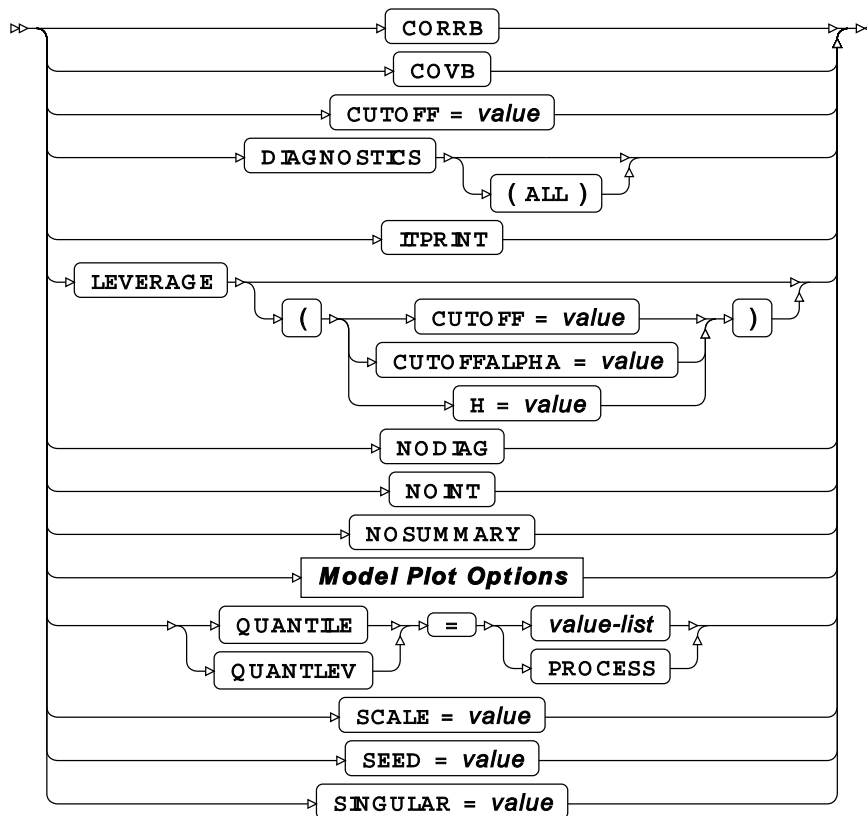
effect-definition



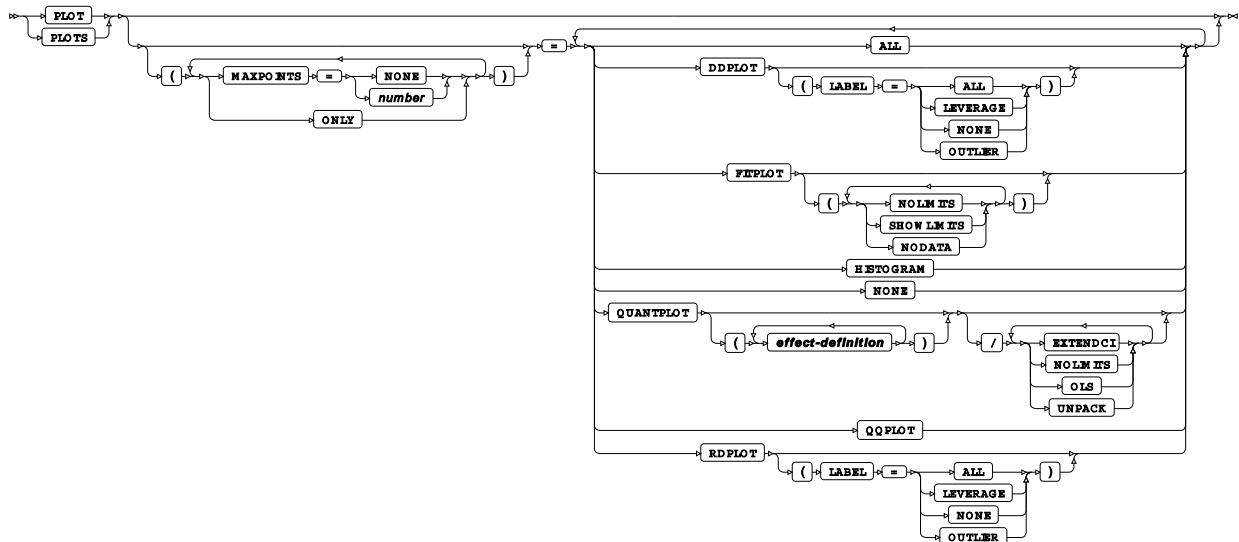
effect-variable



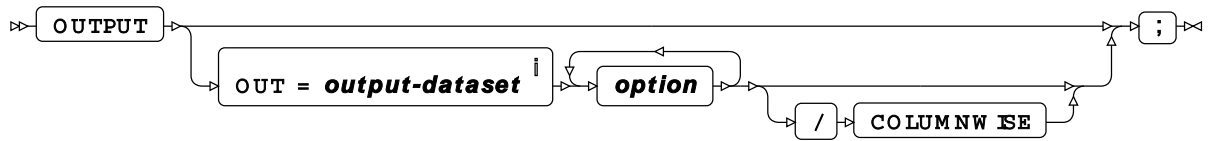
options



Model Plot Options

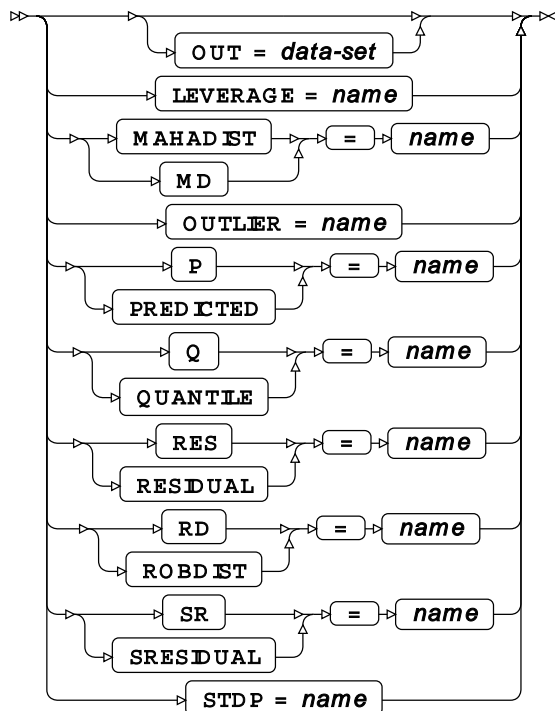


OUTPUT

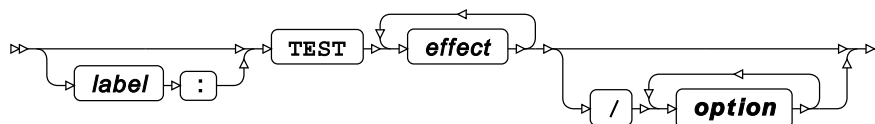


ⁱ See *Output dataset* [↗](#) (page 16).

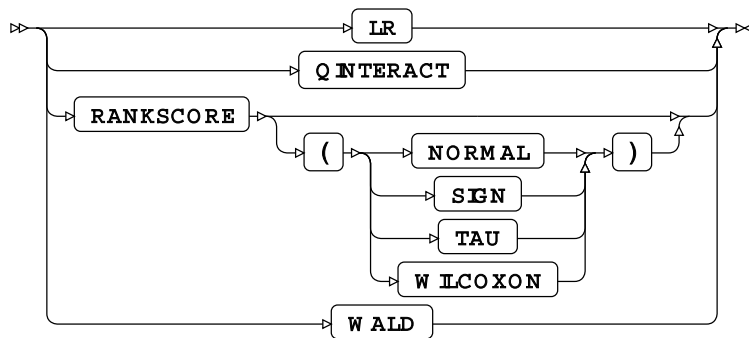
option



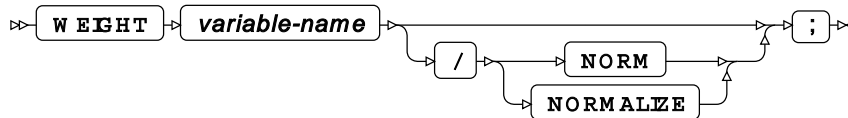
TEST



option



WEIGHT



WHERE



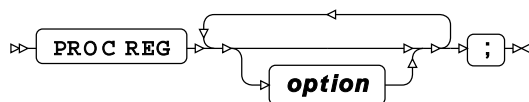
REG procedure

Supported statements

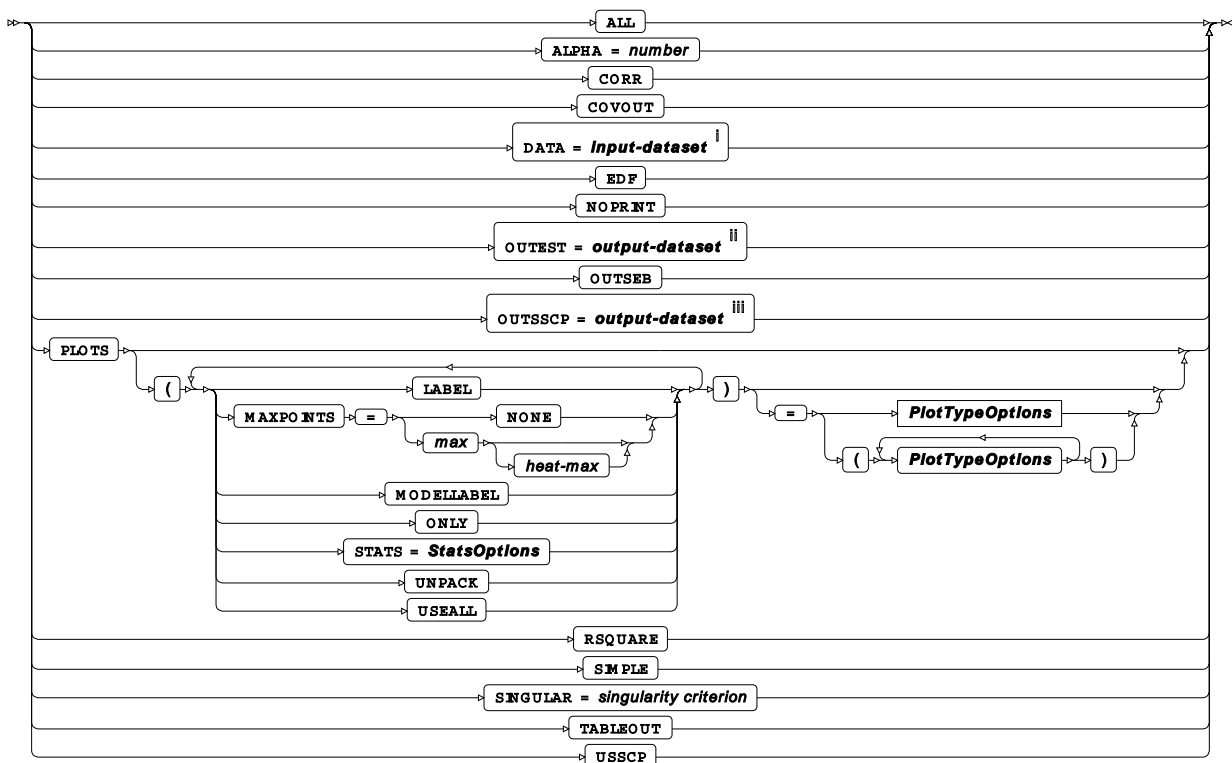
- *PROC REG* [↗](#) (page 3094)
- *ADD* [↗](#) (page 3097)
- *ATTRIB* [↗](#) (page 3097)
- *BY* [↗](#) (page 3097)
- *CODE* [↗](#) (page 3097)
- *DELETE* [↗](#) (page 3098)
- *FORMAT* [↗](#) (page 3098)
- *FREQ* [↗](#) (page 3098)
- *ID* [↗](#) (page 3099)
- *INFORMAT* [↗](#) (page 3099)

- *LABEL* [↗](#) (page 3099)
- *MODEL* [↗](#) (page 3099)
- *MTEST* [↗](#) (page 3101)
- *OUTPUT* [↗](#) (page 3101)
- *PRINT* [↗](#) (page 3102)
- *REFIT* [↗](#) (page 3103)
- *REWEIGHT* [↗](#) (page 3104)
- *TEST* [↗](#) (page 3104)
- *VAR* [↗](#) (page 3104)
- *WEIGHT* [↗](#) (page 3104)
- *WHERE* [↗](#) (page 3104)

PROC REG

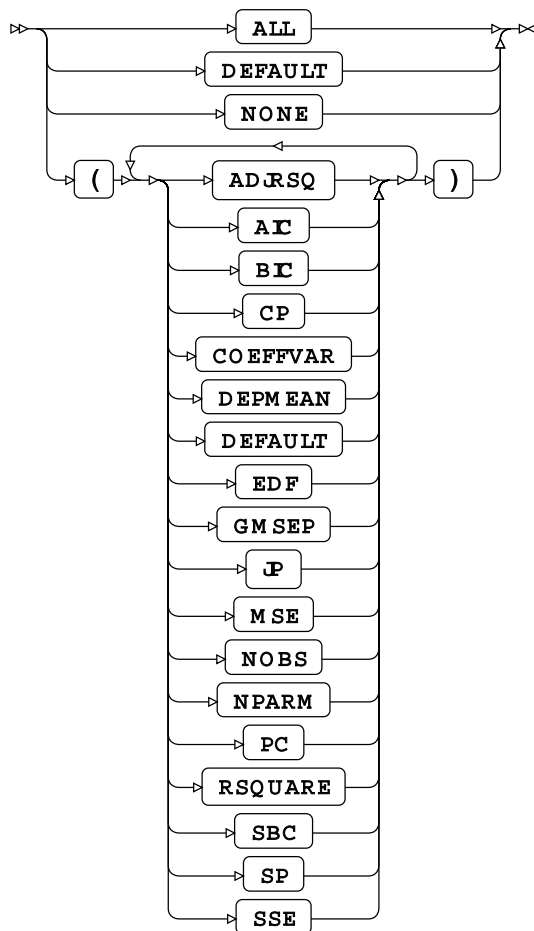


option



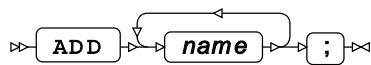
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Output dataset* [↗](#) (page 16).

StatsOptions

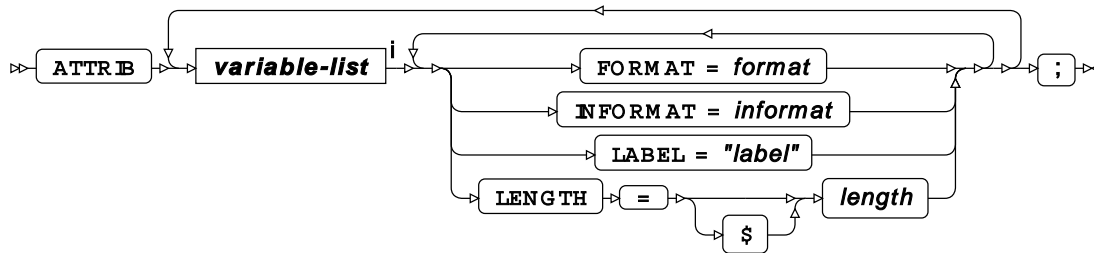


[illegible]

ADD

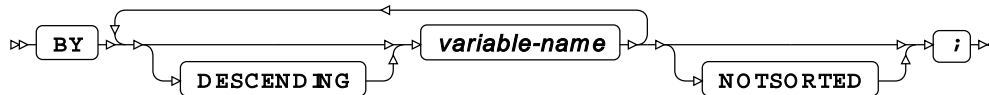


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

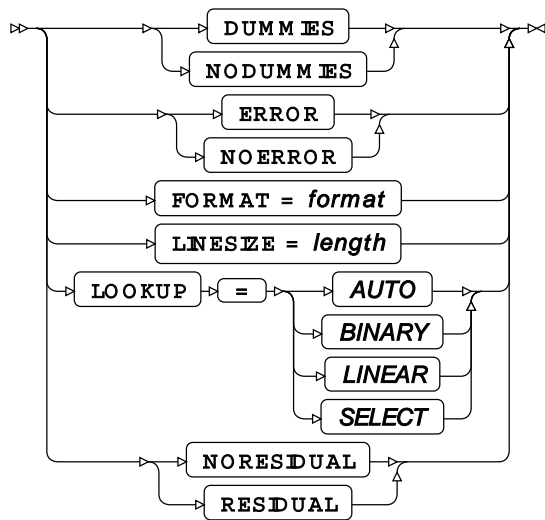
BY



CODE



option



DELETE

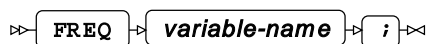


FORMAT

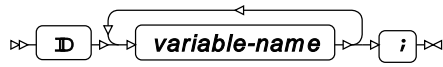


ⁱ See [Variable Lists](#) (page 32).

FREQ



ID

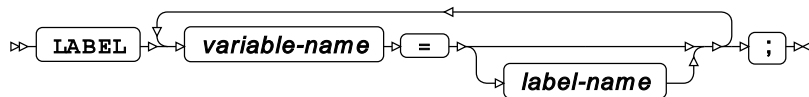


INFORMAT

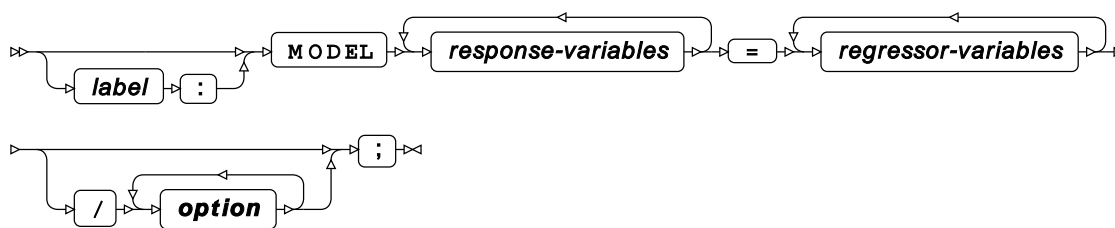


ⁱ See [Variable Lists](#) (page 32).

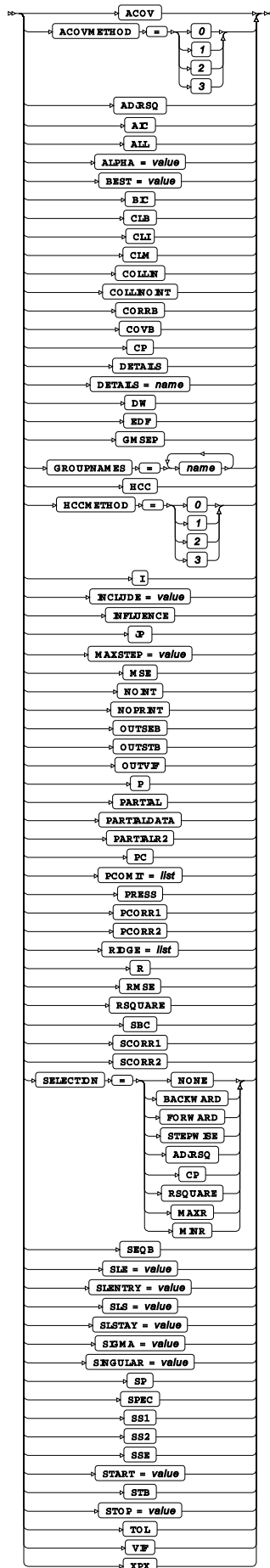
LABEL



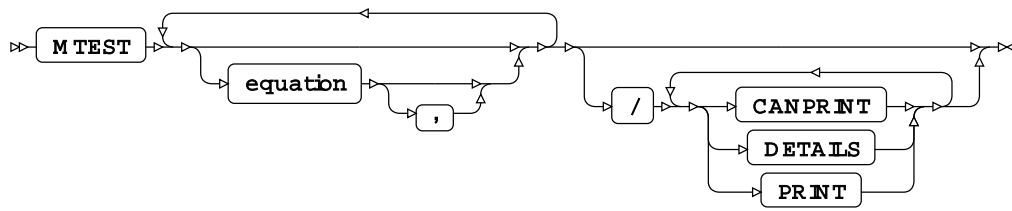
MODEL



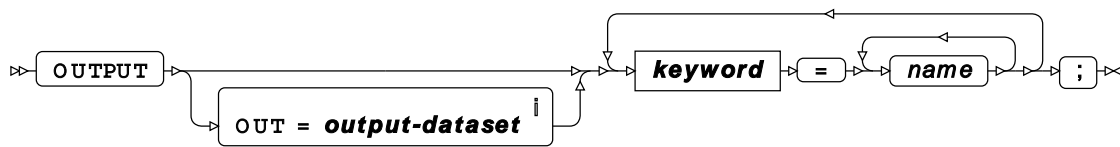
option



MTEST

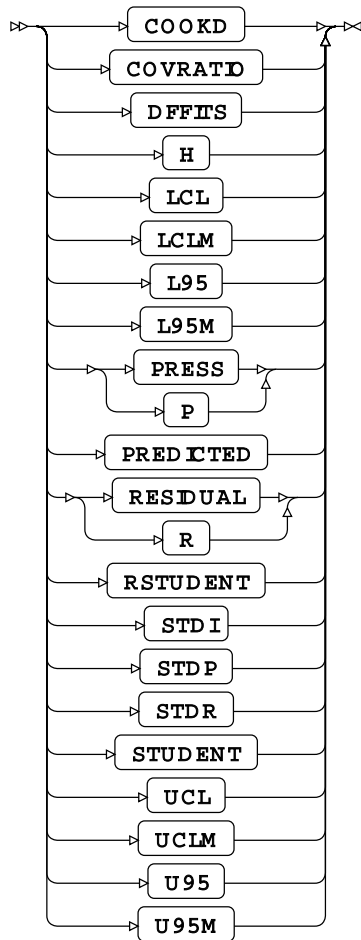


OUTPUT

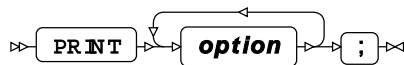


ⁱ See *Output dataset* [↗](#) (page 16).

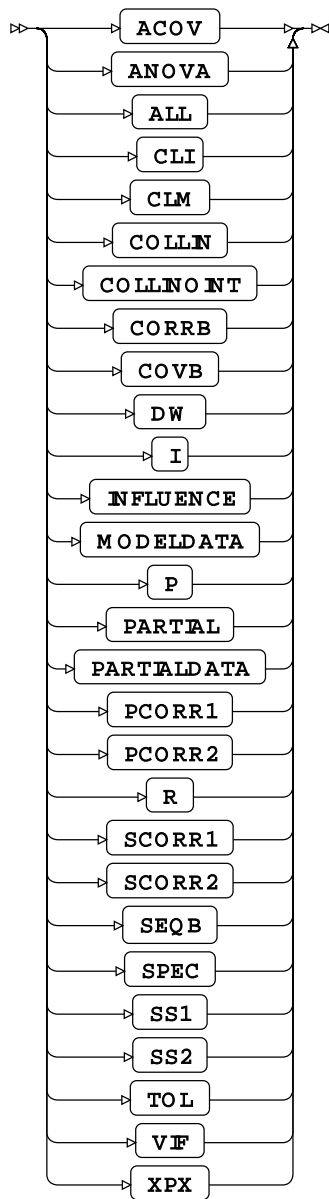
keyword



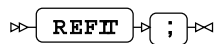
PRINT



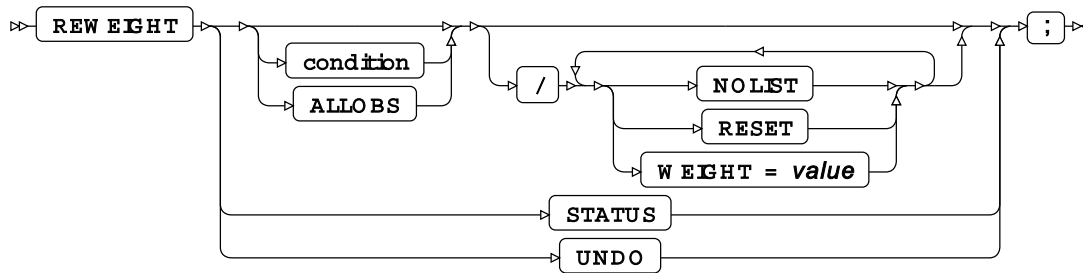
option



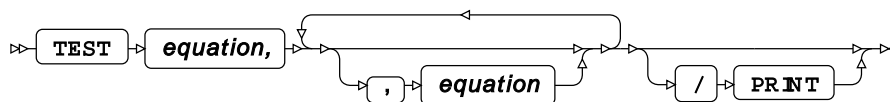
REFIT



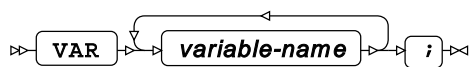
REWEIGHT



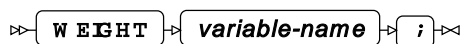
TEST



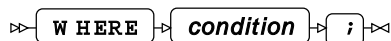
VAR



WEIGHT



WHERE



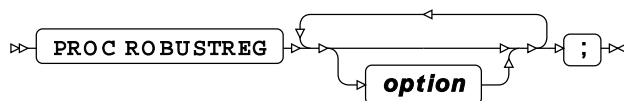
ROBUSTREG procedure

Supported statements

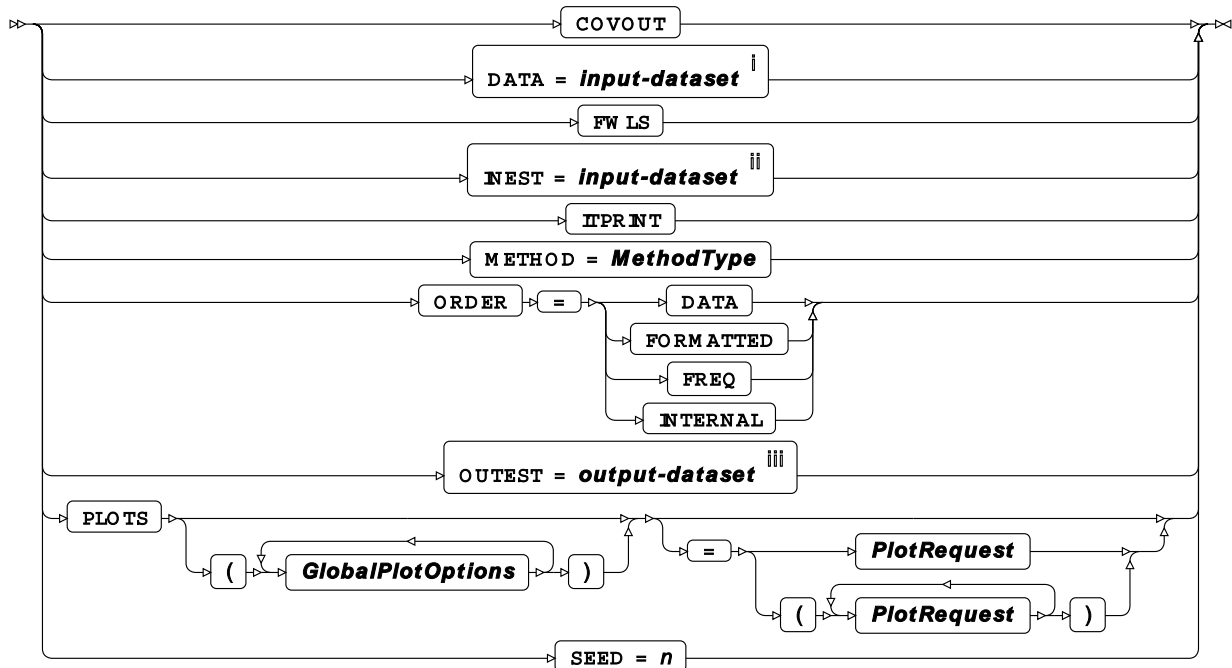
- *PROC ROBUSTREG* [↗](#) (page 3105)

- [ATTRIB](#) (page 3109)
- [BY](#) (page 3109)
- [CLASS](#) (page 3110)
- [FORMAT](#) (page 3110)
- [MODEL](#) (page 3110)
- [ID](#) (page 3111)
- [INFORMAT](#) (page 3112)
- [LABEL](#) (page 3112)
- [OUTPUT](#) (page 3112)
- [TEST](#) (page 3113)
- [WEIGHT](#) (page 3113)
- [WHERE](#) (page 3113)

PROC ROBUSTREG

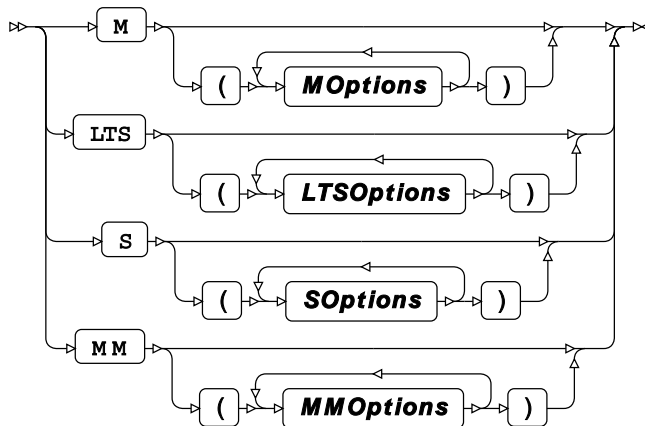


option

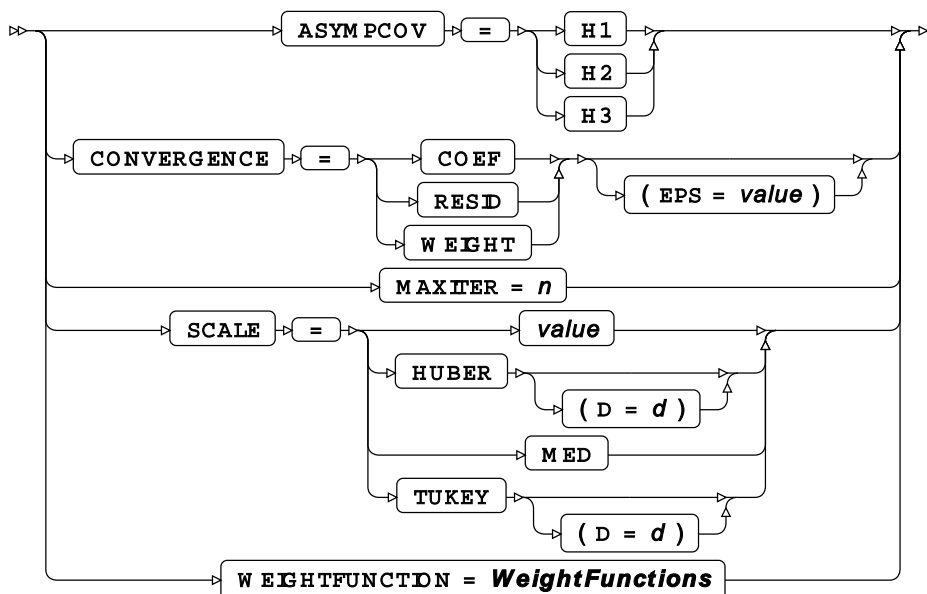


- i See *Input dataset* [↗](#) (page 16).
- ii See *Input dataset* [↗](#) (page 16).
- iii See *Output dataset* [↗](#) (page 16).

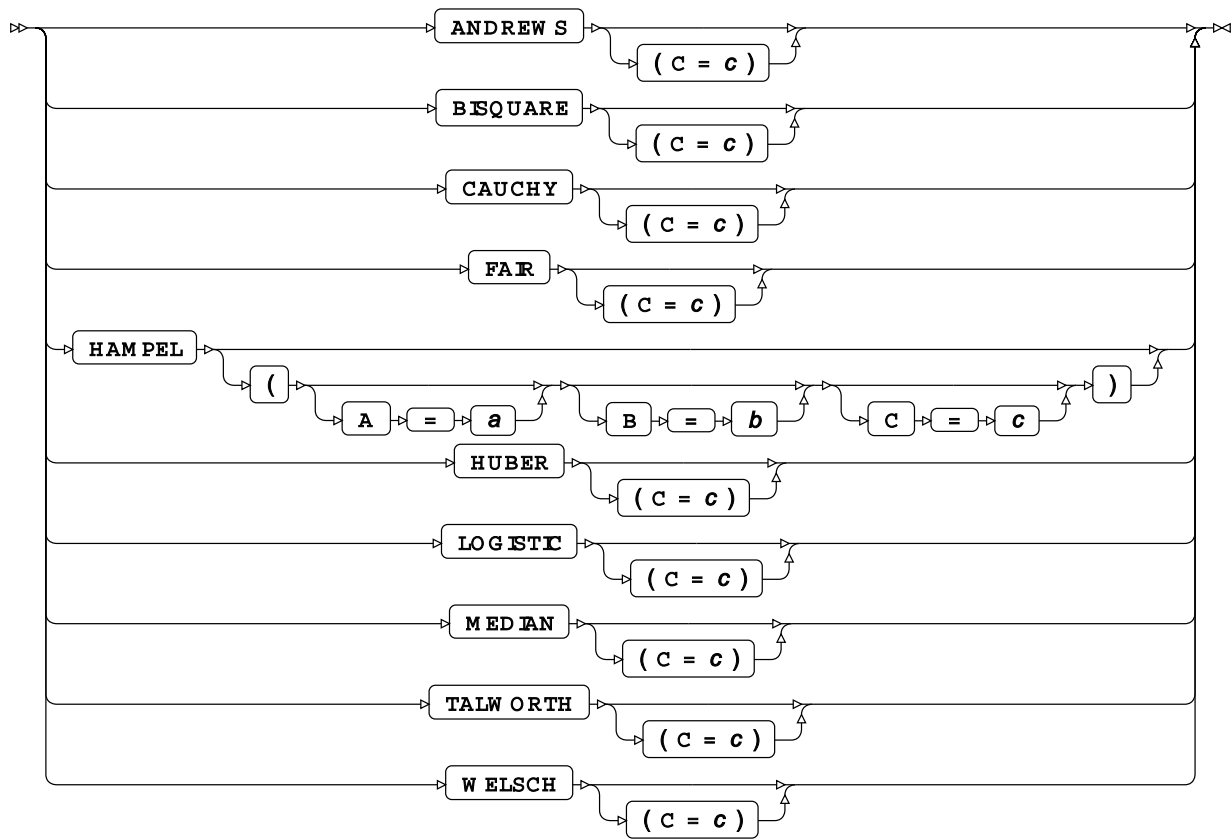
MethodType



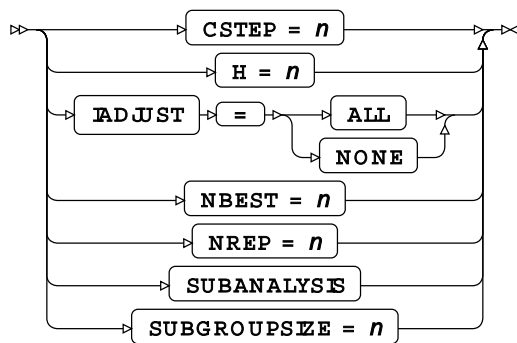
MOptions



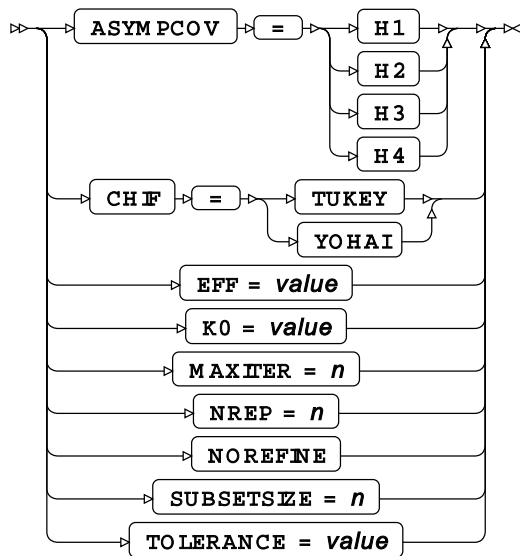
WeightFunctions



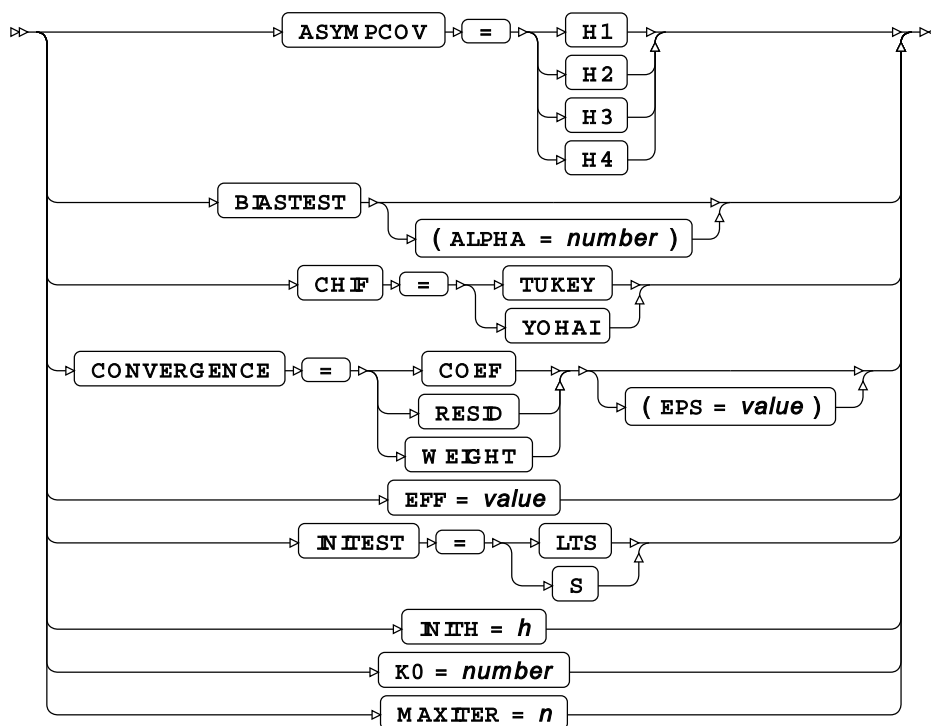
LTSOptions



SOptions



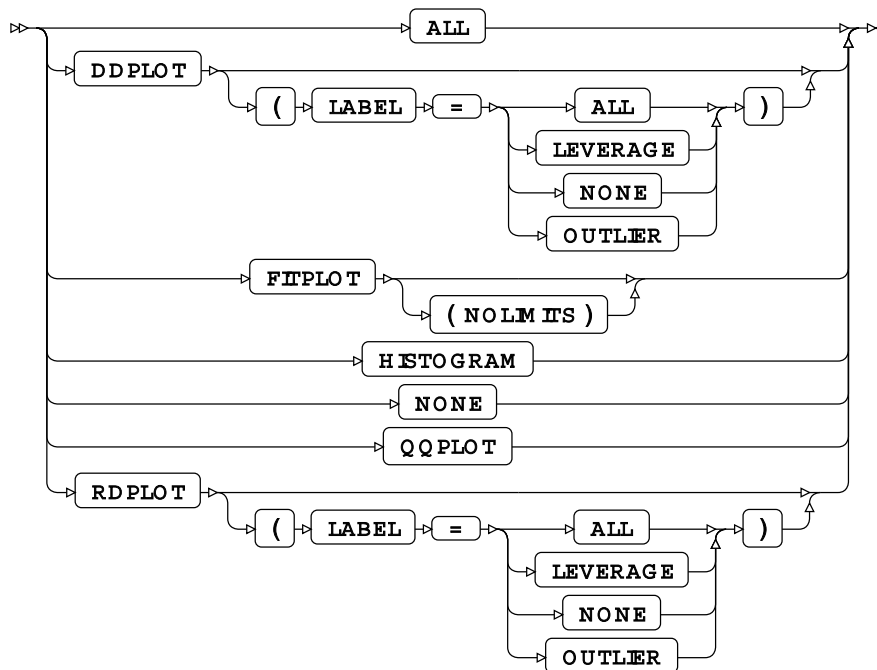
MMOptions



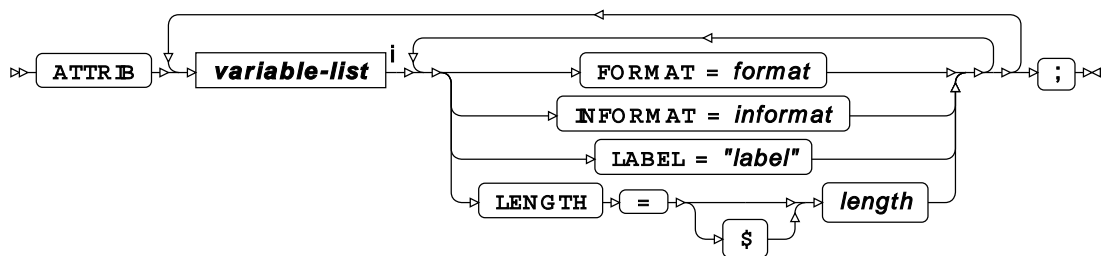
GlobalPlotOptions



PlotRequest

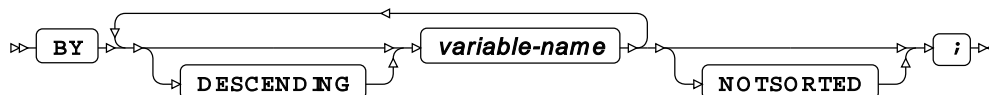


ATTRIB

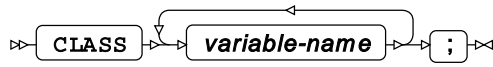


ⁱ See [Variable Lists](#) (page 32).

BY



CLASS

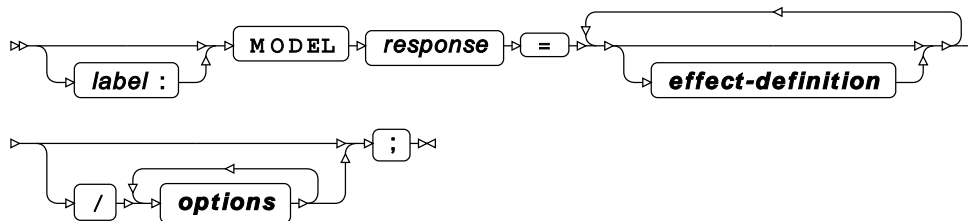


FORMAT

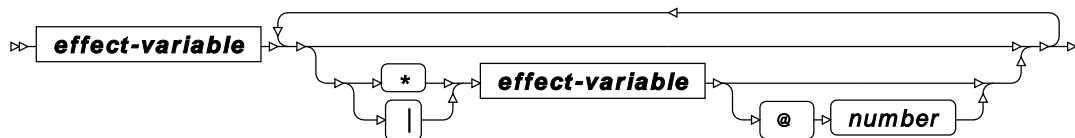


ⁱ See *Variable Lists* [↗](#) (page 32).

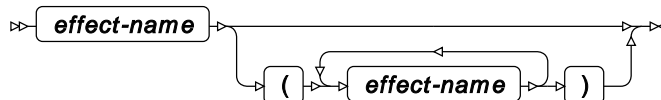
MODEL



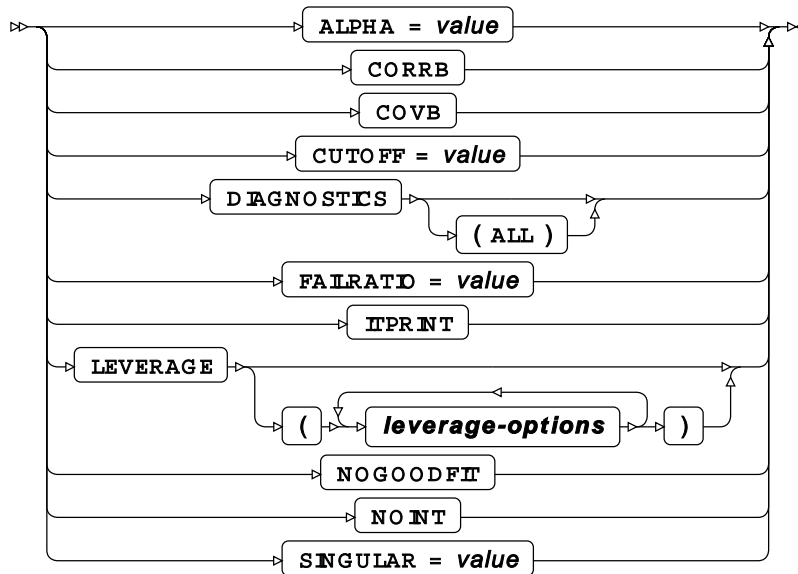
effect-definition



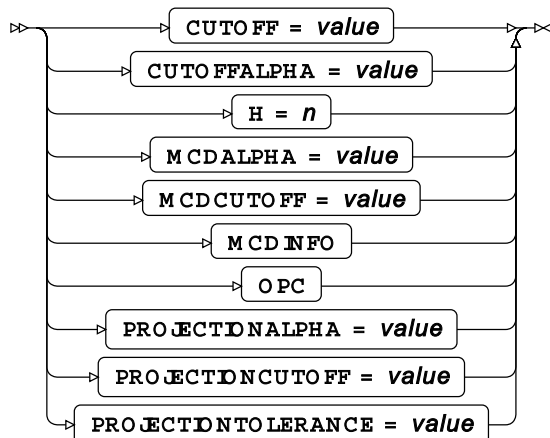
effect-variable



options



leverage-options



ID

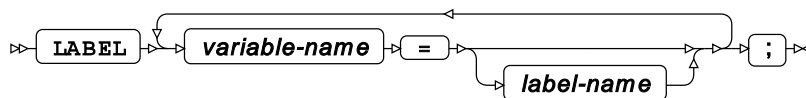


INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL

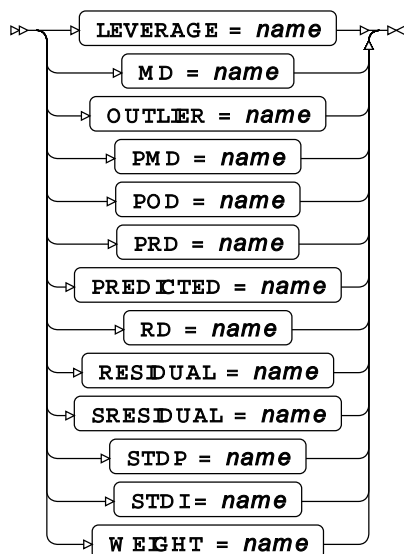


OUTPUT

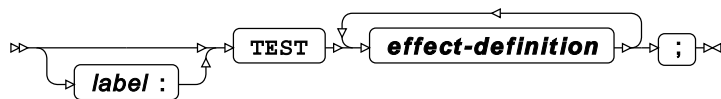


ⁱ See *Output dataset* [↗](#) (page 16).

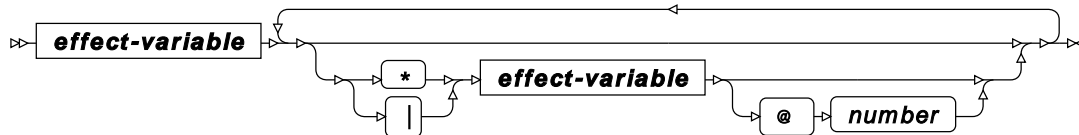
option



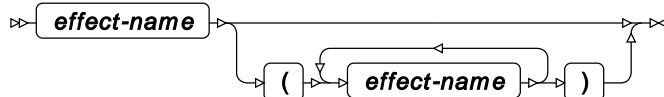
TEST



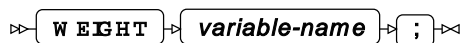
effect-definition



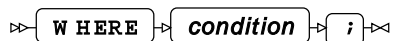
effect-variable



WEIGHT



WHERE



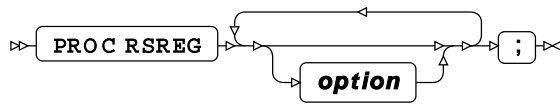
RSREG procedure

Supported statements

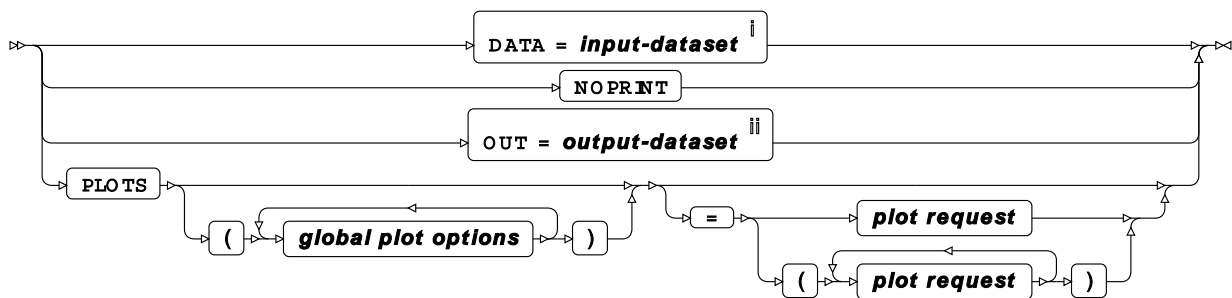
- *PROC RSREG* [↗](#) (page 3114)
- *ATTRIB* [↗](#) (page 3116)
- *BY* [↗](#) (page 3116)
- *FORMAT* [↗](#) (page 3116)
- *FREQ* [↗](#) (page 3116)

- *ID* [↗](#) (page 3116)
- *INFORMAT* [↗](#) (page 3117)
- *LABEL* [↗](#) (page 3117)
- *MODEL* [↗](#) (page 3117)
- *RIDGE* [↗](#) (page 3118)
- *WEIGHT* [↗](#) (page 3119)
- *WHERE* [↗](#) (page 3119)

PROC RSREG



option



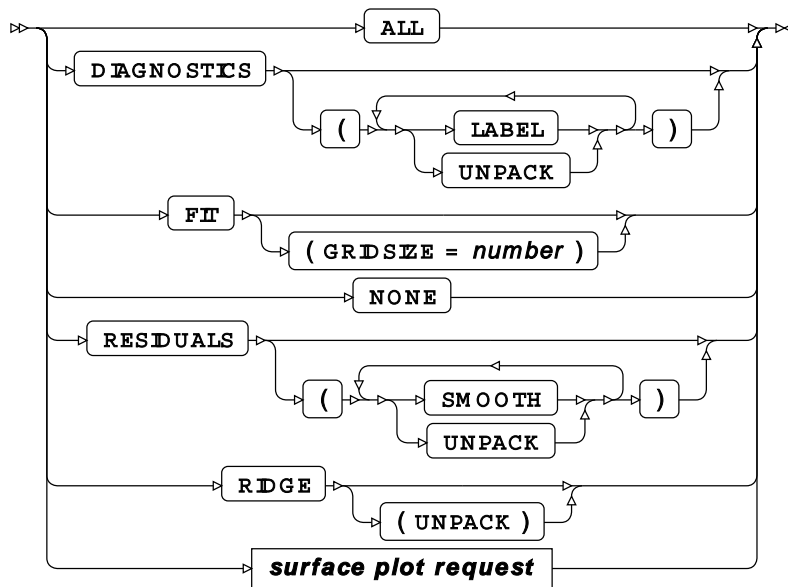
ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

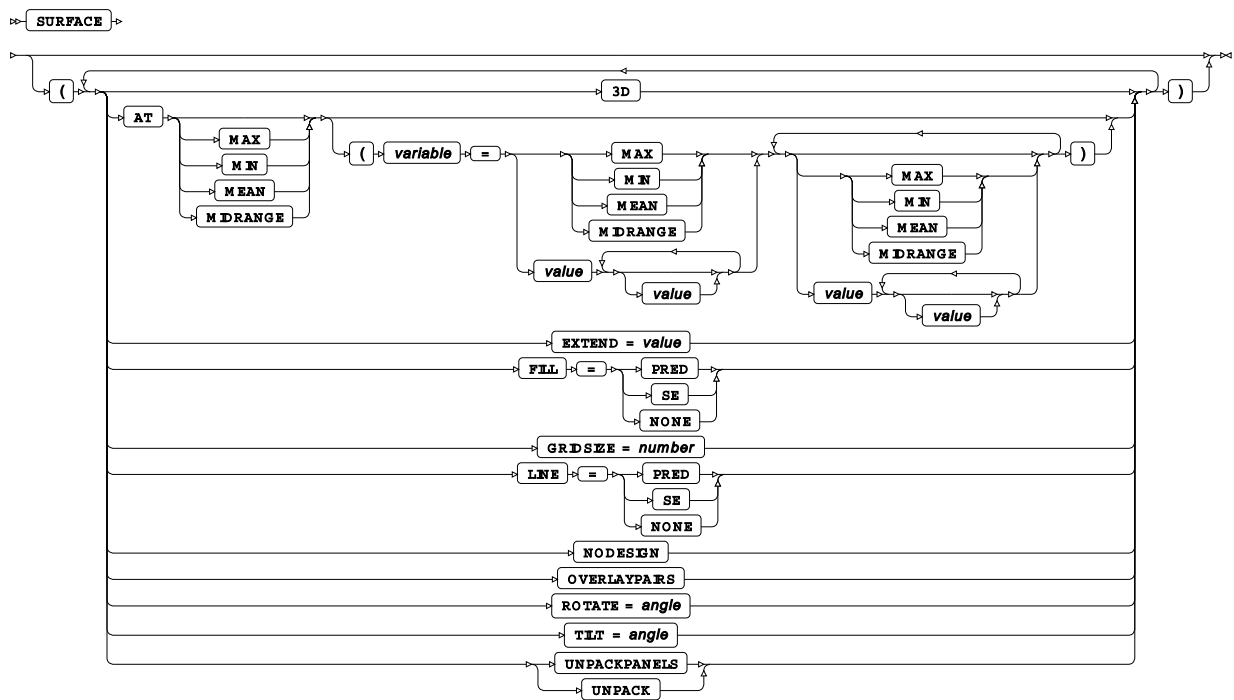
global plot options



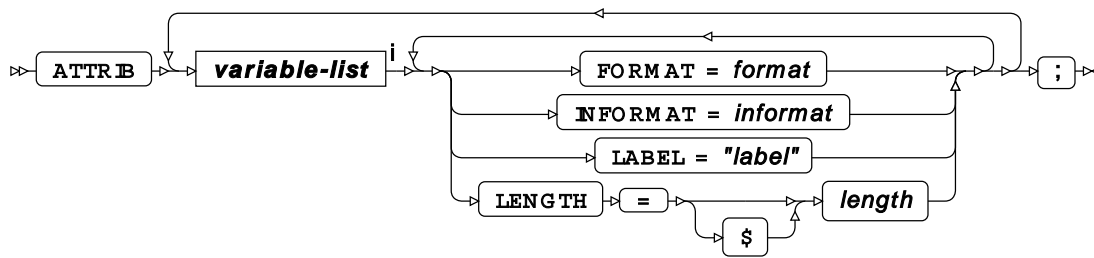
plot request



surface plot request

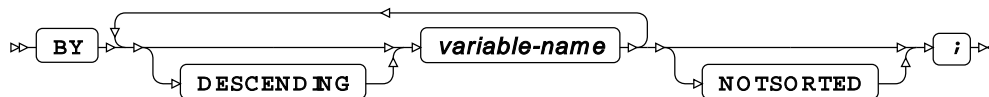


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

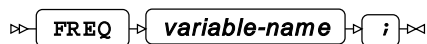


FORMAT

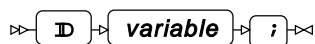


ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ



ID

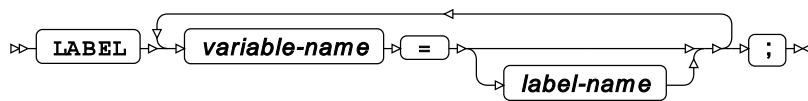


INFORMAT

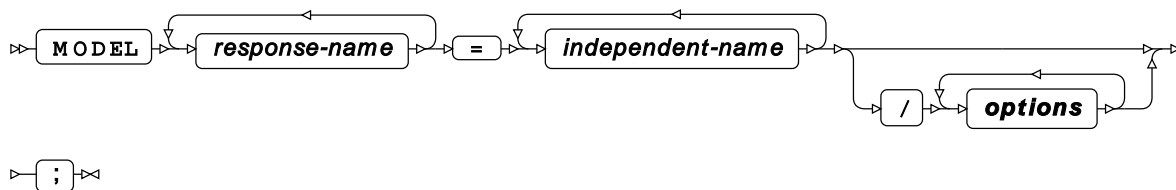


ⁱ See *Variable Lists* [↗](#) (page 32).

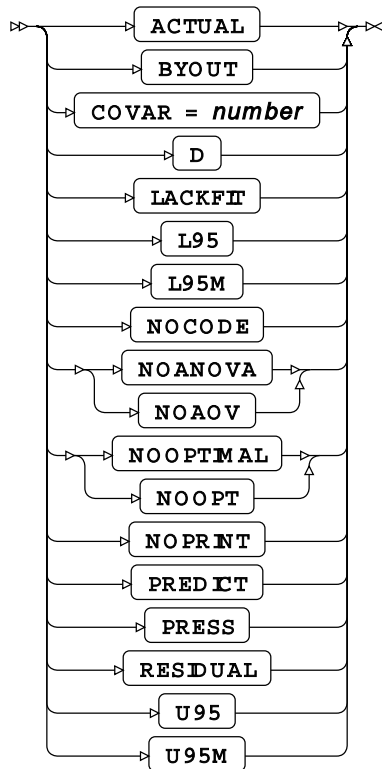
LABEL



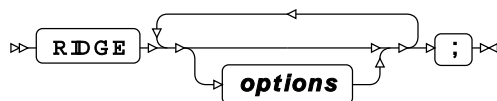
MODEL



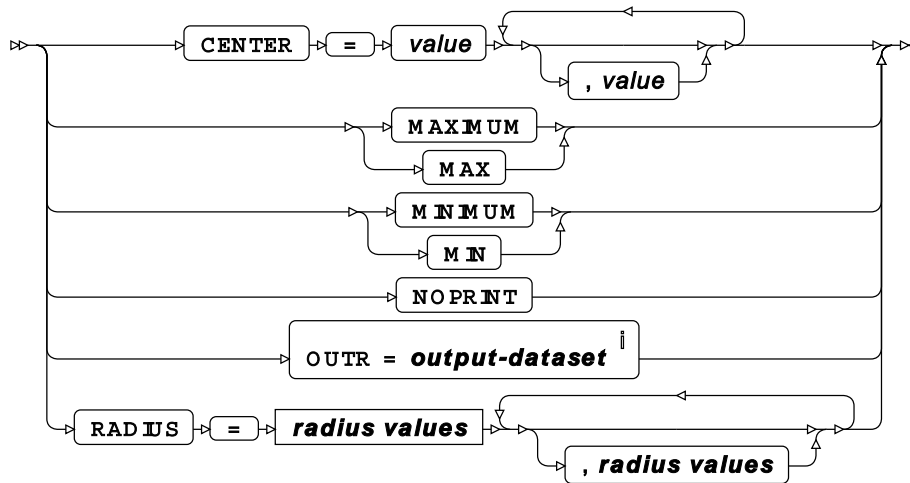
options



RIDGE

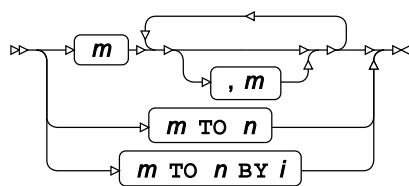


options

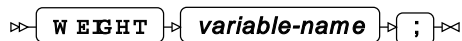


ⁱ See [Output dataset](#) (page 16).

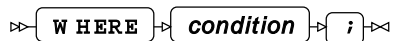
radius values



WEIGHT



WHERE

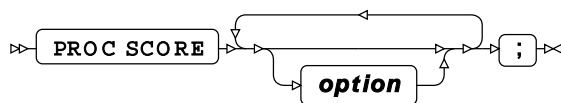


SCORE procedure

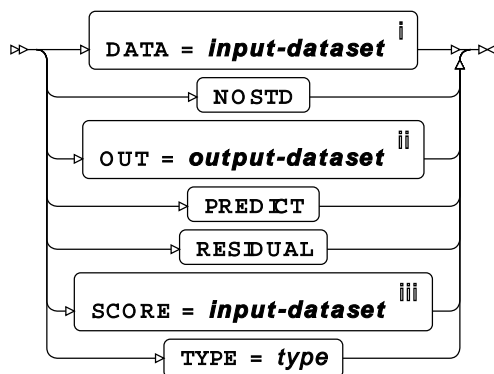
Supported statements

- *PROC SCORE* [↗](#) (page 3120)
- *ATTRIB* [↗](#) (page 3121)
- *BY* [↗](#) (page 3121)
- *FORMAT* [↗](#) (page 3121)
- *ID* [↗](#) (page 3121)
- *INFORMAT* [↗](#) (page 3121)
- *LABEL* [↗](#) (page 3122)
- *VAR* [↗](#) (page 3122)
- *WHERE* [↗](#) (page 3122)

PROC SCORE



option

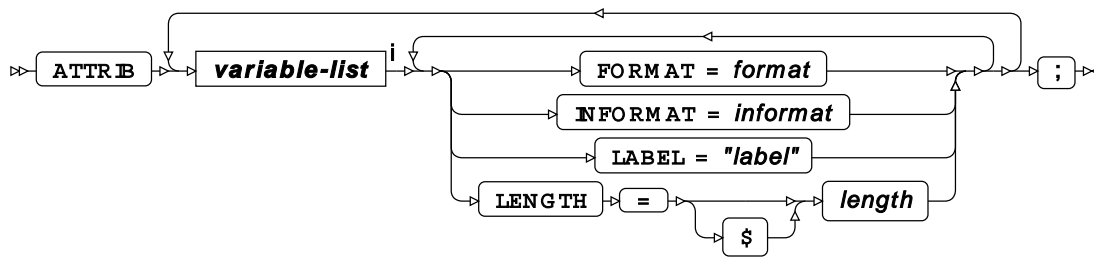


ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Input dataset* [↗](#) (page 16).

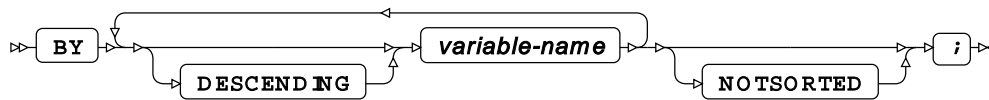
ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

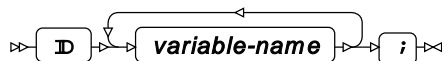


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

ID

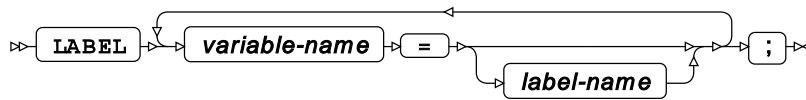


INFORMAT

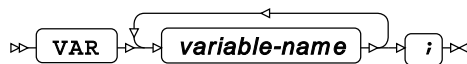


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



VAR



WHERE

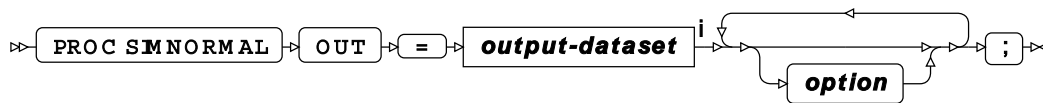


SIMNORMAL procedure

Supported statements

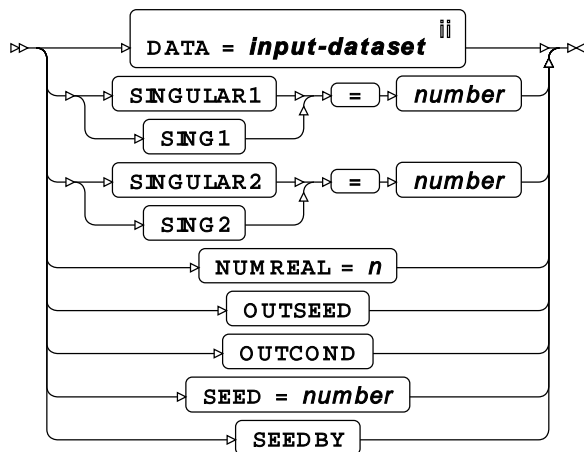
- *PROC SIMNORMAL* [↗](#) (page 3123)
- *ATTRIB* [↗](#) (page 3123)
- *BY* [↗](#) (page 3124)
- *CONDITION* [↗](#) (page 3124)
- *FORMAT* [↗](#) (page 3124)
- *INFORMAT* [↗](#) (page 3124)
- *LABEL* [↗](#) (page 3124)
- *VAR* [↗](#) (page 3125)
- *WHERE* [↗](#) (page 3125)

PROC SIMNORMAL



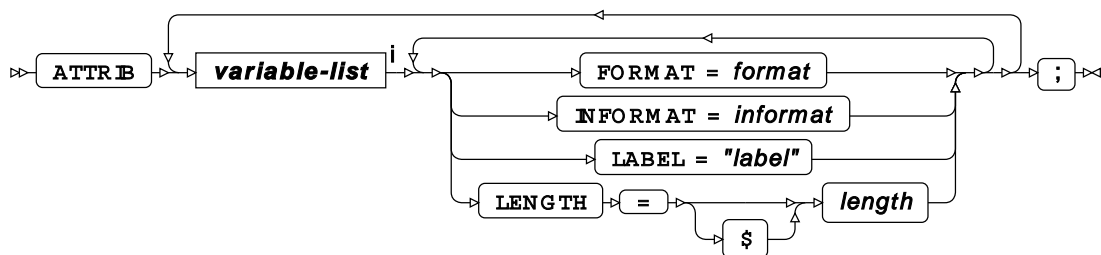
ⁱ See *Input dataset* [↗](#) (page 16).

option



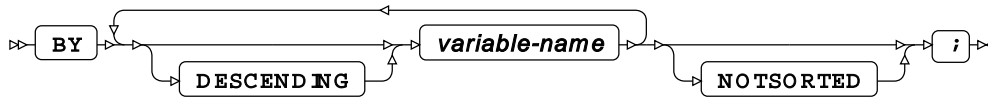
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

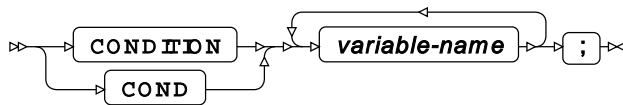


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



CONDITION



FORMAT



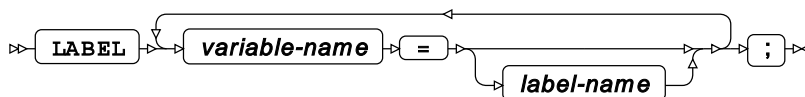
ⁱ See [Variable Lists](#) (page 32).

INFORMAT

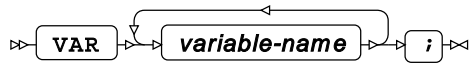


ⁱ See [Variable Lists](#) (page 32).

LABEL



VAR



WHERE

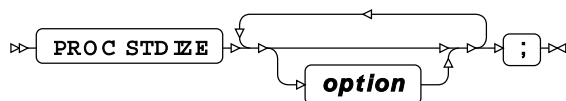


STDIZE procedure

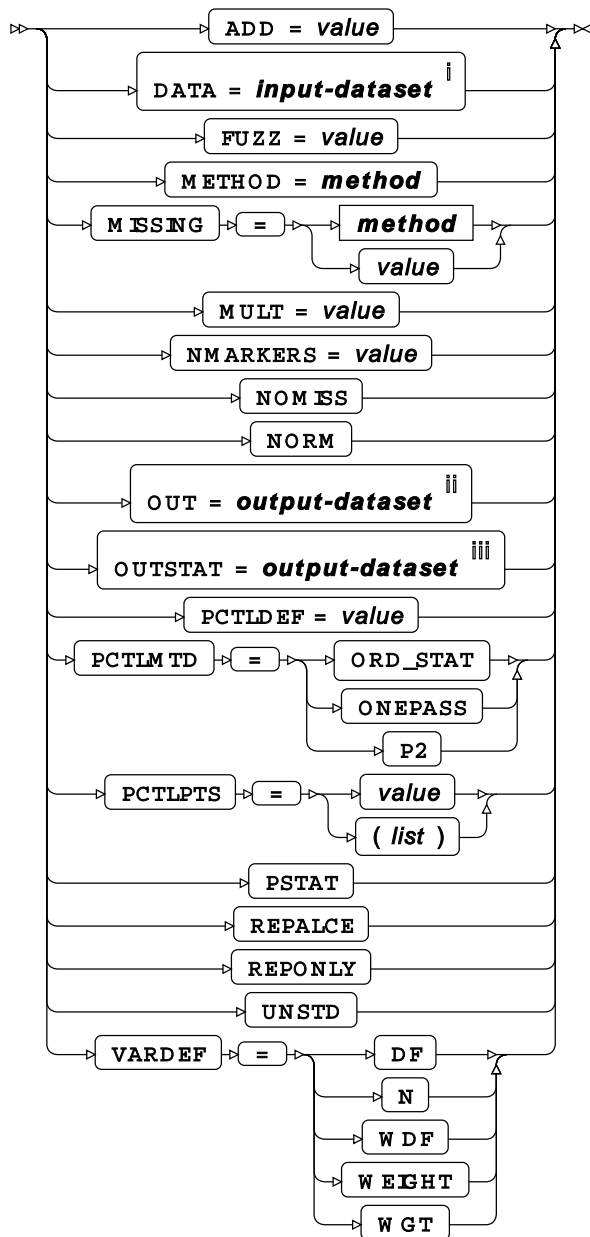
Supported statements

- *PROC STDIZE* [↗](#) (page 3125)
- *ATTRIB* [↗](#) (page 3127)
- *BY* [↗](#) (page 3127)
- *FORMAT* [↗](#) (page 3128)
- *FREQ* [↗](#) (page 3128)
- *INFORMAT* [↗](#) (page 3128)
- *LABEL* [↗](#) (page 3128)
- *LOCATION* [↗](#) (page 3128)
- *SCALE* [↗](#) (page 3128)
- *VAR* [↗](#) (page 3129)
- *WEIGHT* [↗](#) (page 3129)
- *WHERE* [↗](#) (page 3129)

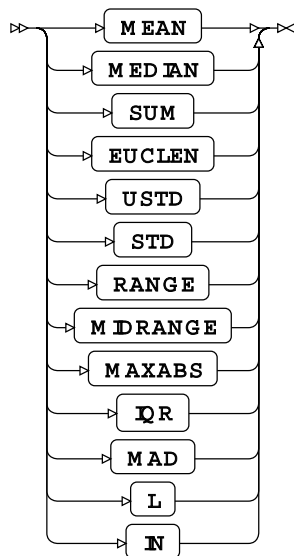
PROC STDIZE



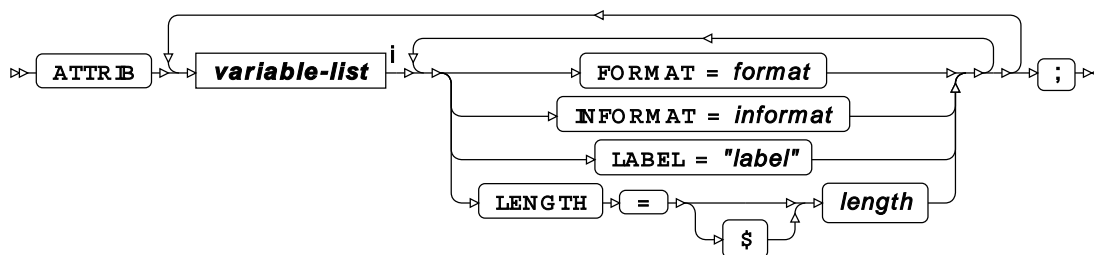
option

ⁱ See *Input dataset* [↗](#) (page 16).ⁱⁱ See *Input dataset* [↗](#) (page 16).ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

method

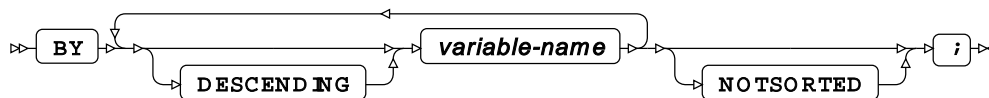


ATTRIB

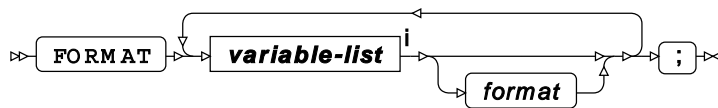


ⁱ See [Variable Lists](#) (page 32).

BY

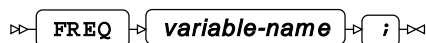


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

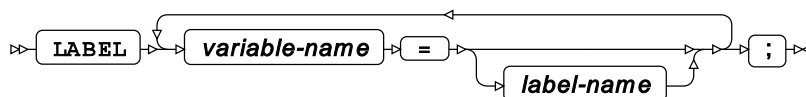


INFORMAT

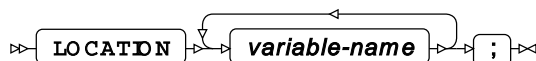


ⁱ See *Variable Lists* [↗](#) (page 32).

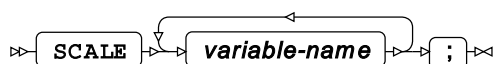
LABEL



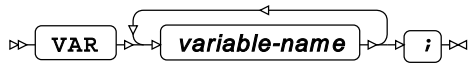
LOCATION



SCALE



VAR



WEIGHT



WHERE

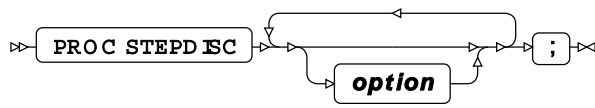


STEPPDISC procedure

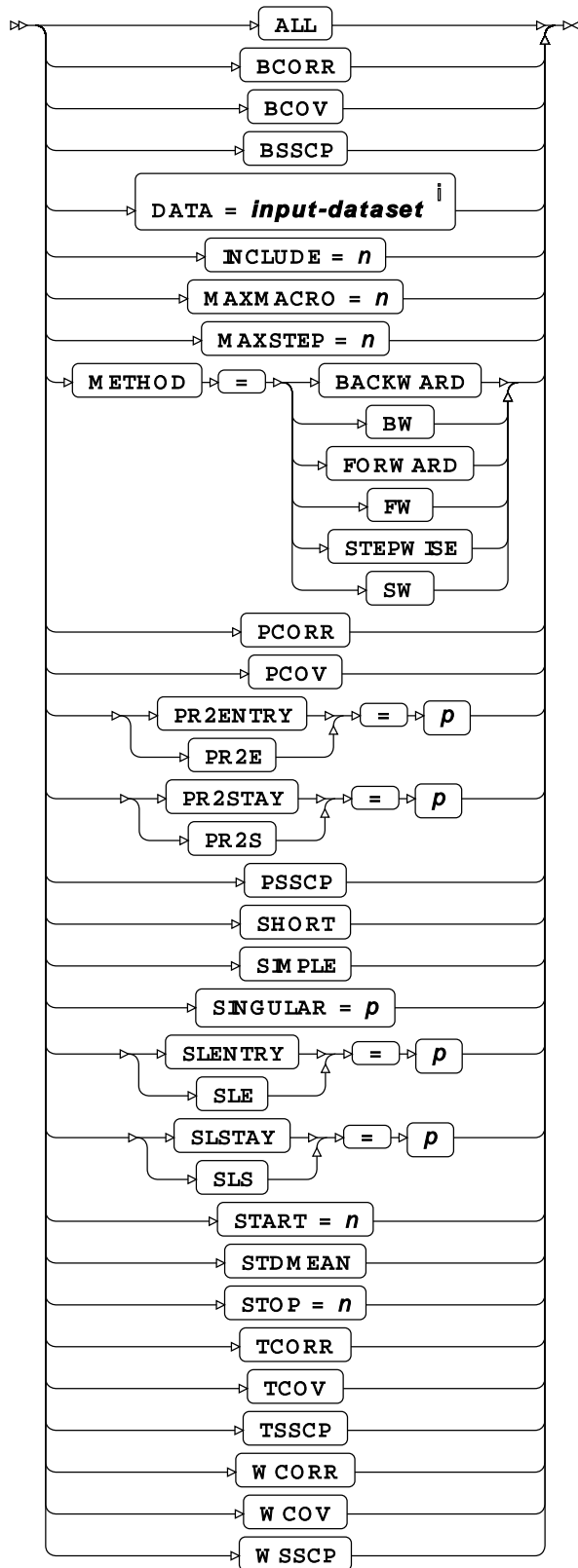
Supported statements

- *PROC STEPPDISC* [↗](#) (page 3130)
- *ATTRIB* [↗](#) (page 3132)
- *BY* [↗](#) (page 3132)
- *CLASS* [↗](#) (page 3132)
- *FORMAT* [↗](#) (page 3132)
- *FREQ* [↗](#) (page 3133)
- *INFORMAT* [↗](#) (page 3133)
- *LABEL* [↗](#) (page 3133)
- *VAR* [↗](#) (page 3133)
- *WEIGHT* [↗](#) (page 3133)
- *WHERE* [↗](#) (page 3133)

PROC STEPDISC

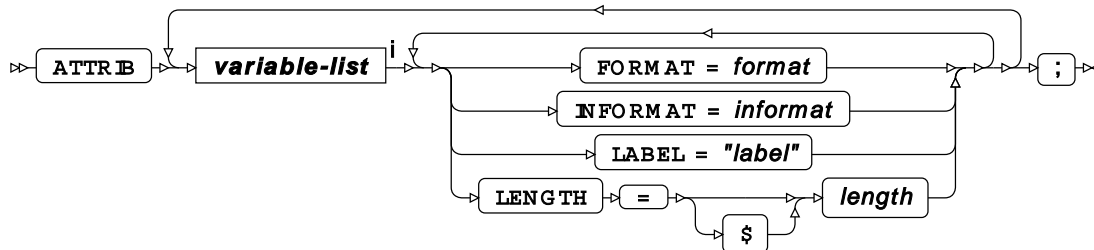


option



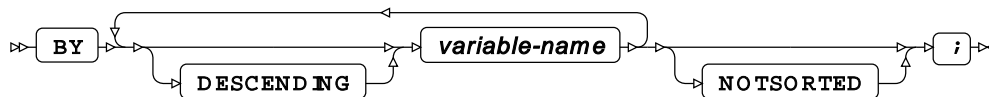
ⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

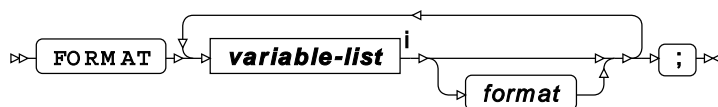
BY



CLASS



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

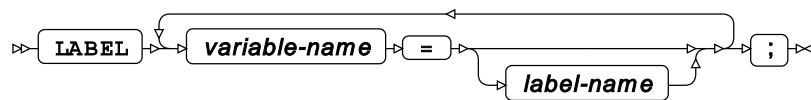


INFORMAT

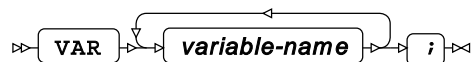


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



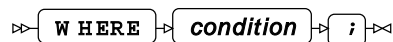
VAR



WEIGHT



WHERE

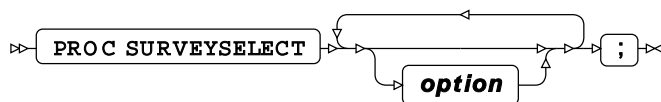


SURVEYSELECT procedure

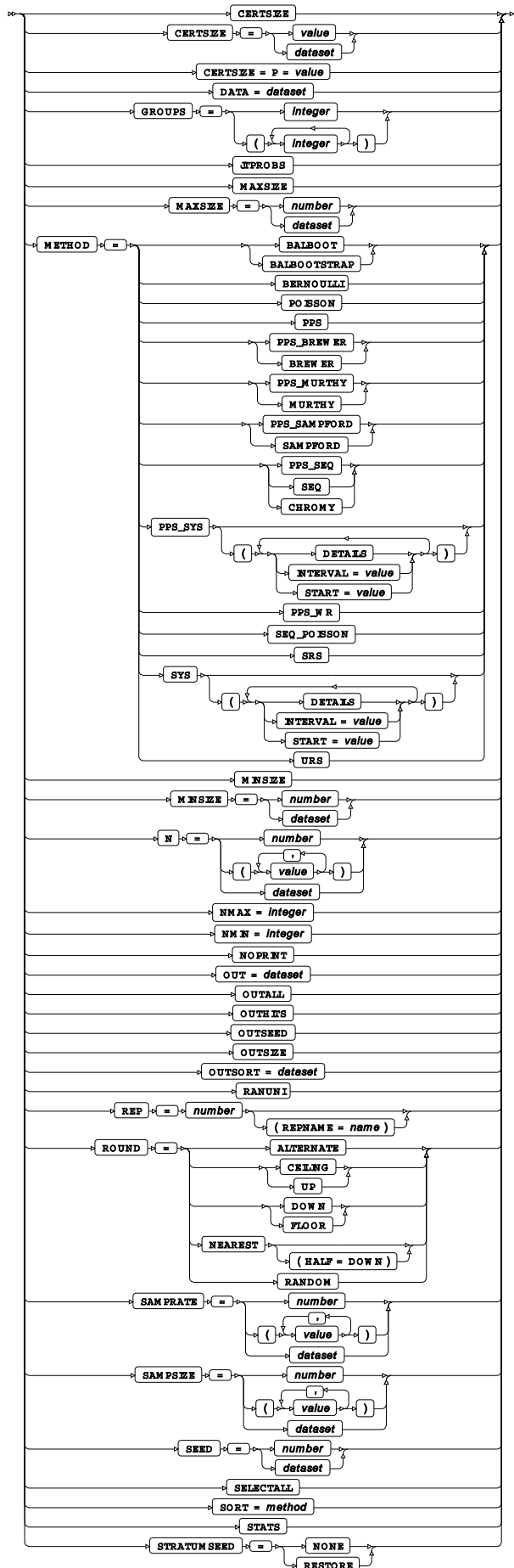
Supported statements

- *PROC SURVEYSELECT* [↗](#) (page 3134)
- *ATTRIB* [↗](#) (page 3136)
- *CONTROL* [↗](#) (page 3136)
- *FORMAT* [↗](#) (page 3136)
- *FREQ* [↗](#) (page 3136)
- *ID* [↗](#) (page 3136)
- *INFORMAT* [↗](#) (page 3137)
- *LABEL* [↗](#) (page 3137)
- *SIZE* [↗](#) (page 3137)
- *STRATA* [↗](#) (page 3137)
- *WHERE* [↗](#) (page 3138)

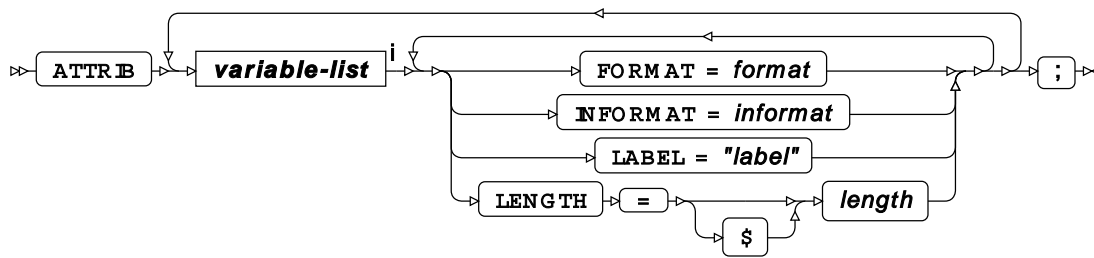
PROC SURVEYSELECT



option

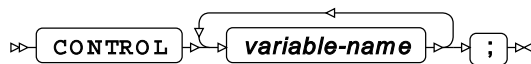


ATTRIB



ⁱ See [Variable Lists](#) (page 32).

CONTROL

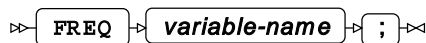


FORMAT

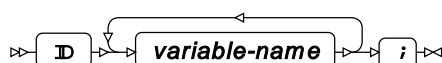


ⁱ See [Variable Lists](#) (page 32).

FREQ



ID

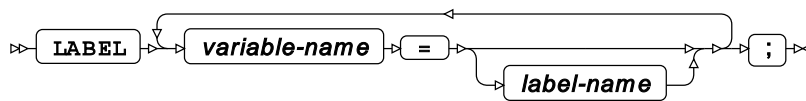


INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

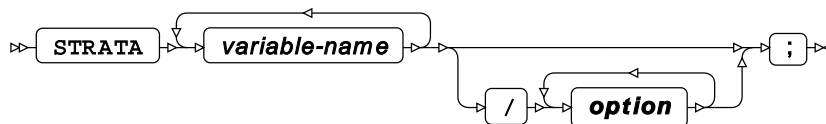
LABEL



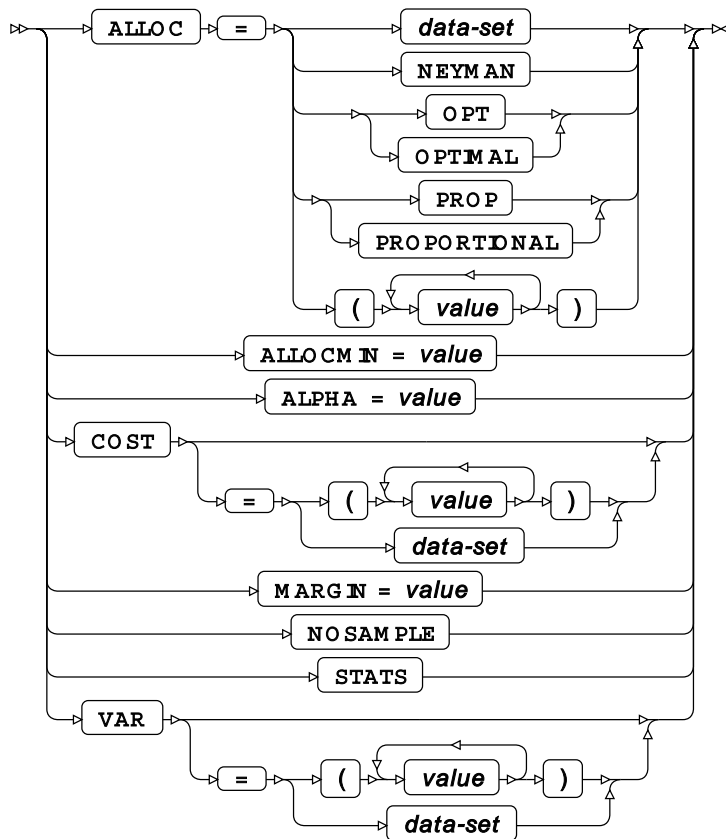
SIZE



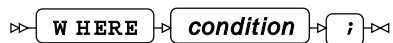
STRATA



option



WHERE



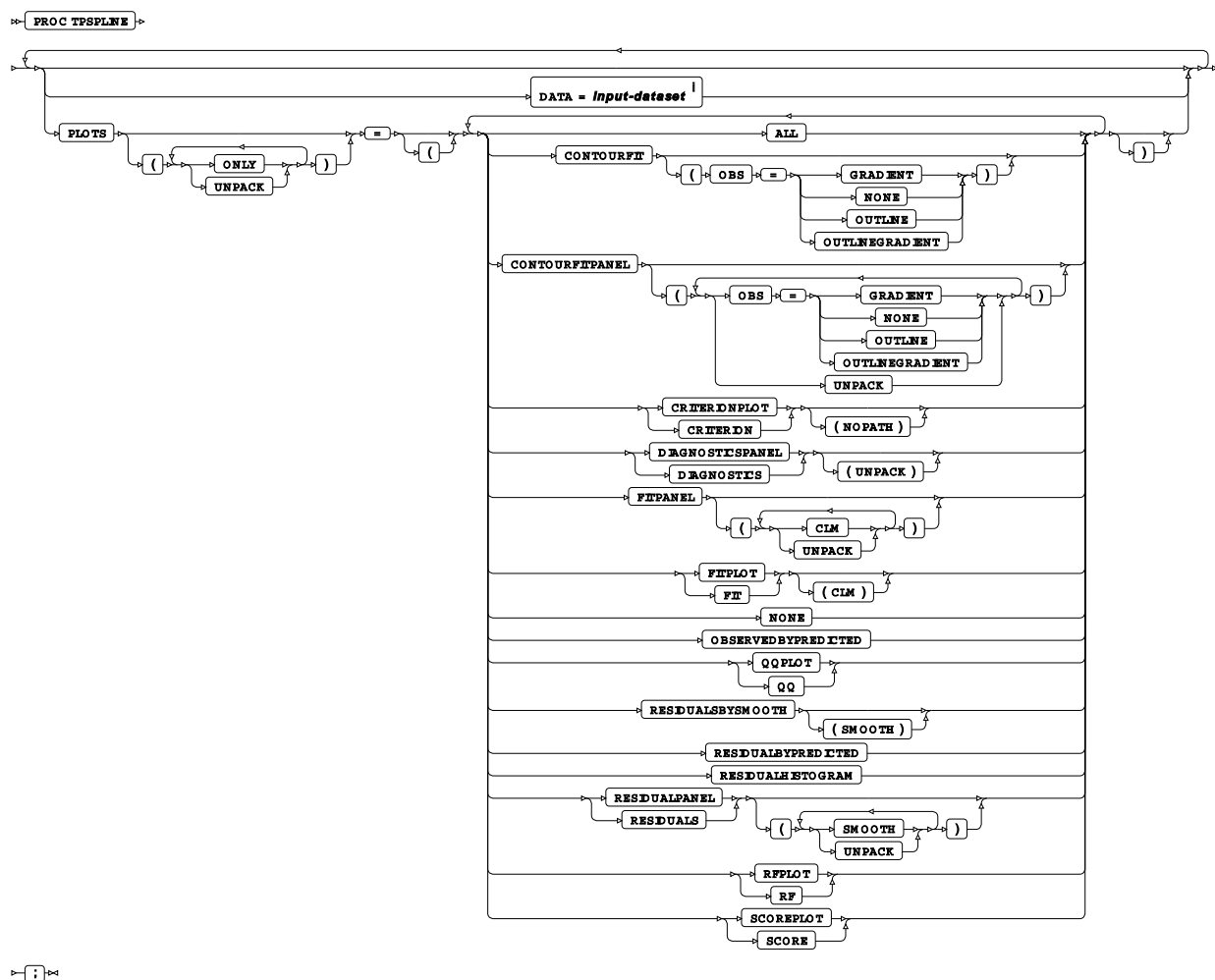
TPSPLINE procedure

Supported statements

- *PROC TPSPLINE* [↗](#) (page 3139)
- *ATTRIB* [↗](#) (page 3140)
- *BY* [↗](#) (page 3140)
- *FORMAT* [↗](#) (page 3140)
- *ID* [↗](#) (page 3140)
- *INFORMAT* [↗](#) (page 3140)

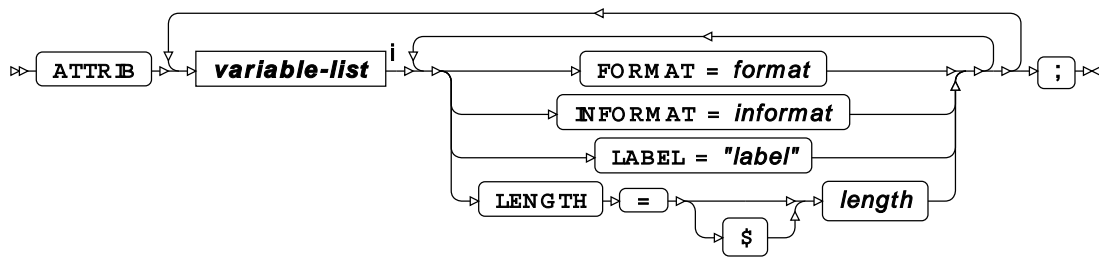
- [FREQ](#) (page 3141)
- [LABEL](#) (page 3141)
- [MODEL](#) (page 3141)
- [OUTPUT](#) (page 3142)
- [SCORE](#) (page 3142)
- [WHERE](#) (page 3142)

PROC TPSPLINE



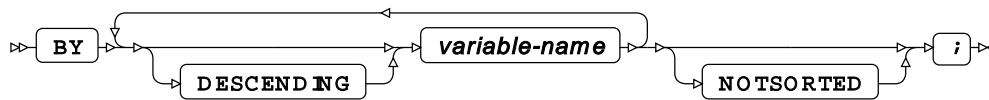
ⁱ See [Input dataset](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

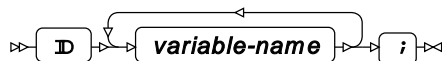


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

ID



INFORMAT

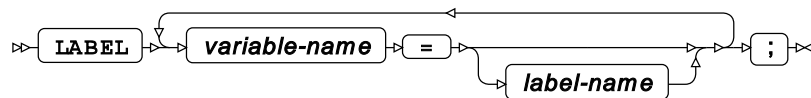


ⁱ See [Variable Lists](#) (page 32).

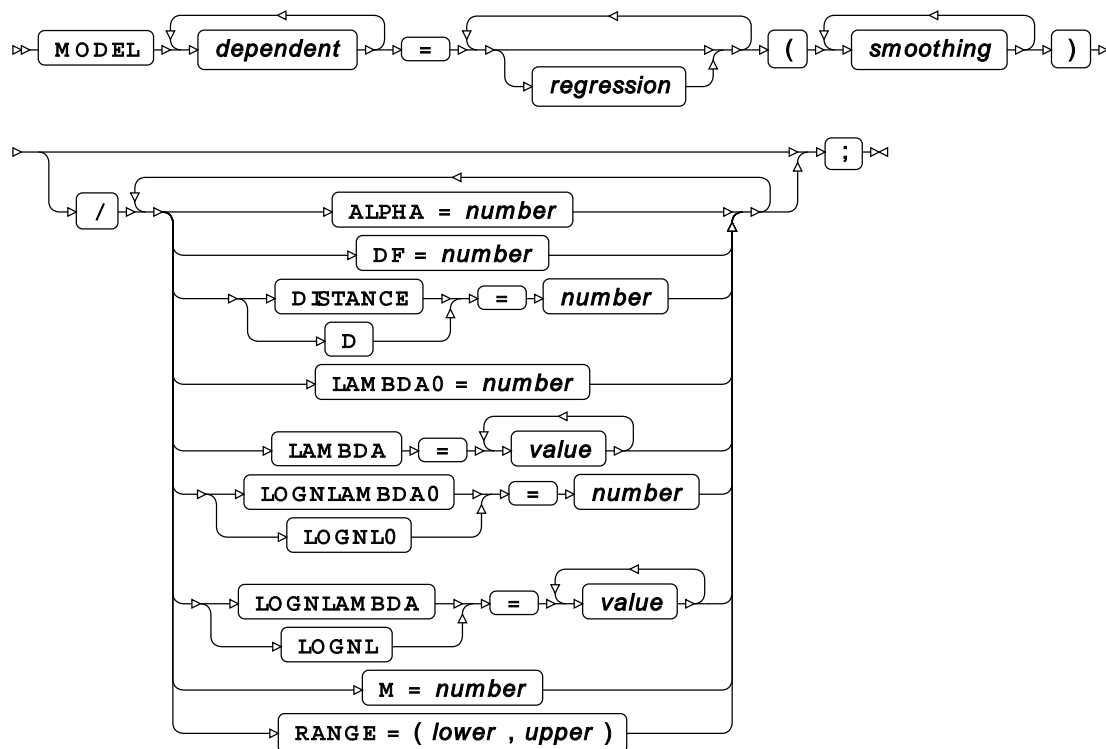
FREQ



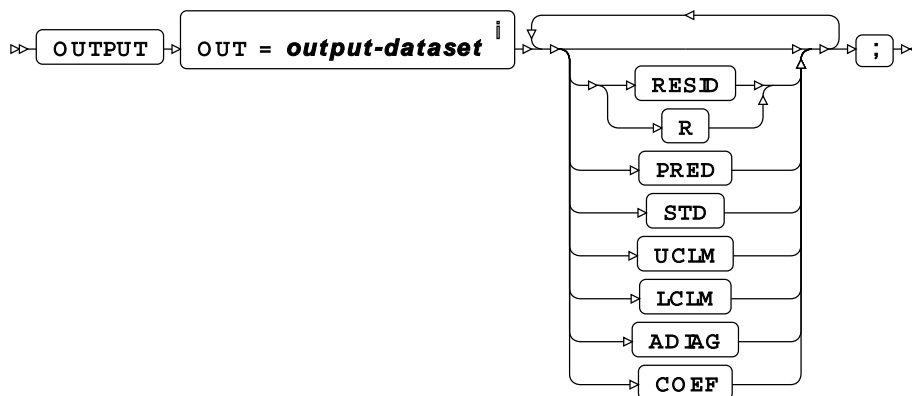
LABEL



MODEL

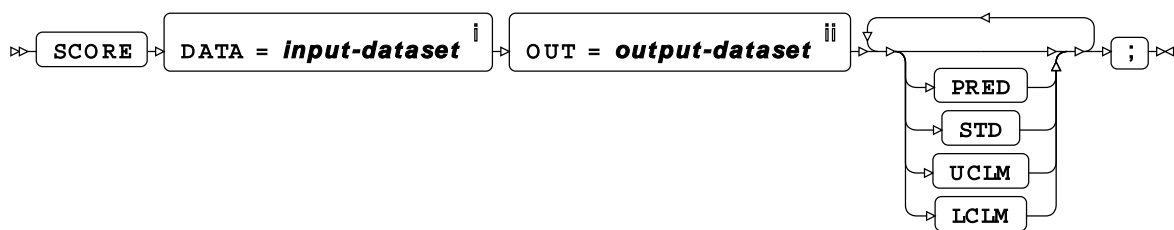


OUTPUT



ⁱ See *Output dataset* [↗](#) (page 16).

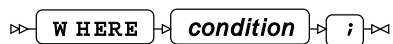
SCORE



ⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

WHERE



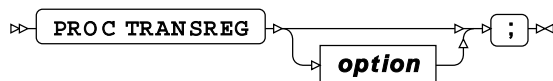
TRANSREG procedure

Supported statements

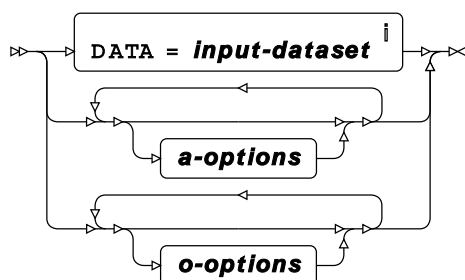
- *PROC TRANSREG* [↗](#) (page 3143)
- *ATTRIB* [↗](#) (page 3146)

- [BY](#) (page 3146)
- [ID](#) (page 3146)
- [FORMAT](#) (page 3146)
- [FREQ](#) (page 3146)
- [INFORMAT](#) (page 3147)
- [LABEL](#) (page 3147)
- [MODEL](#) (page 3147)
- [OUTPUT](#) (page 3151)
- [WEIGHT](#) (page 3153)
- [WHERE](#) (page 3153)

PROC TRANSREG

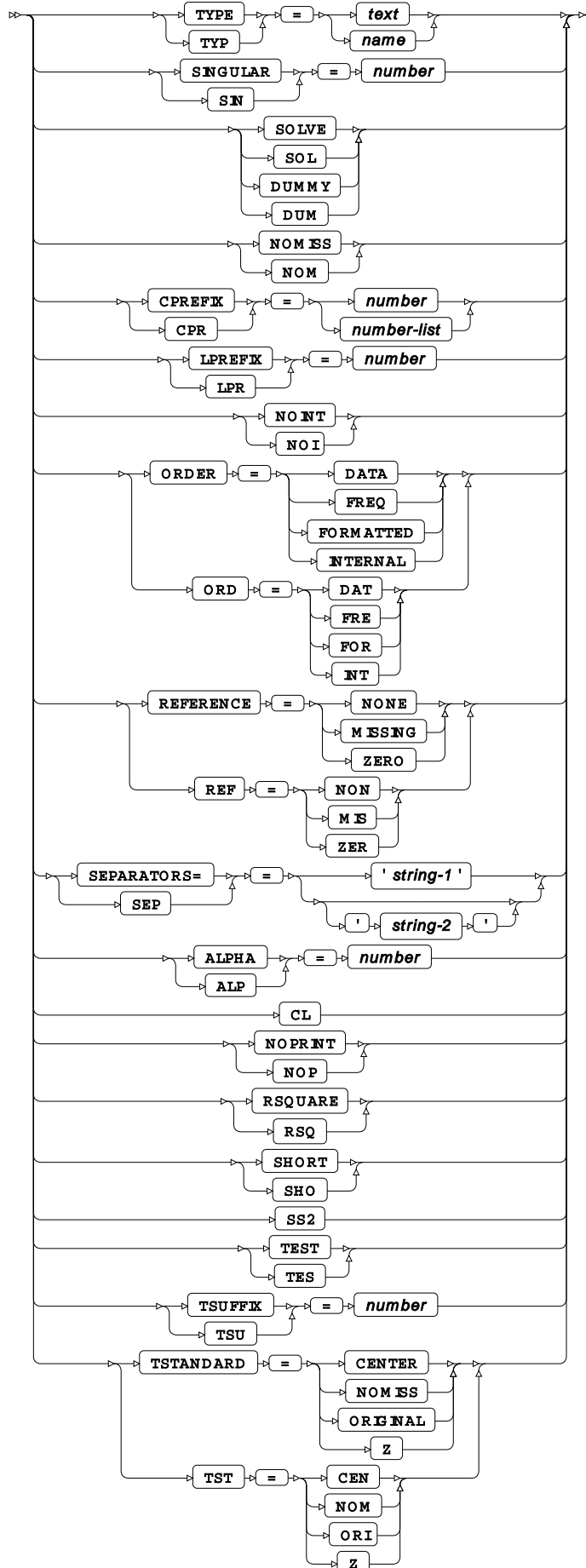


option

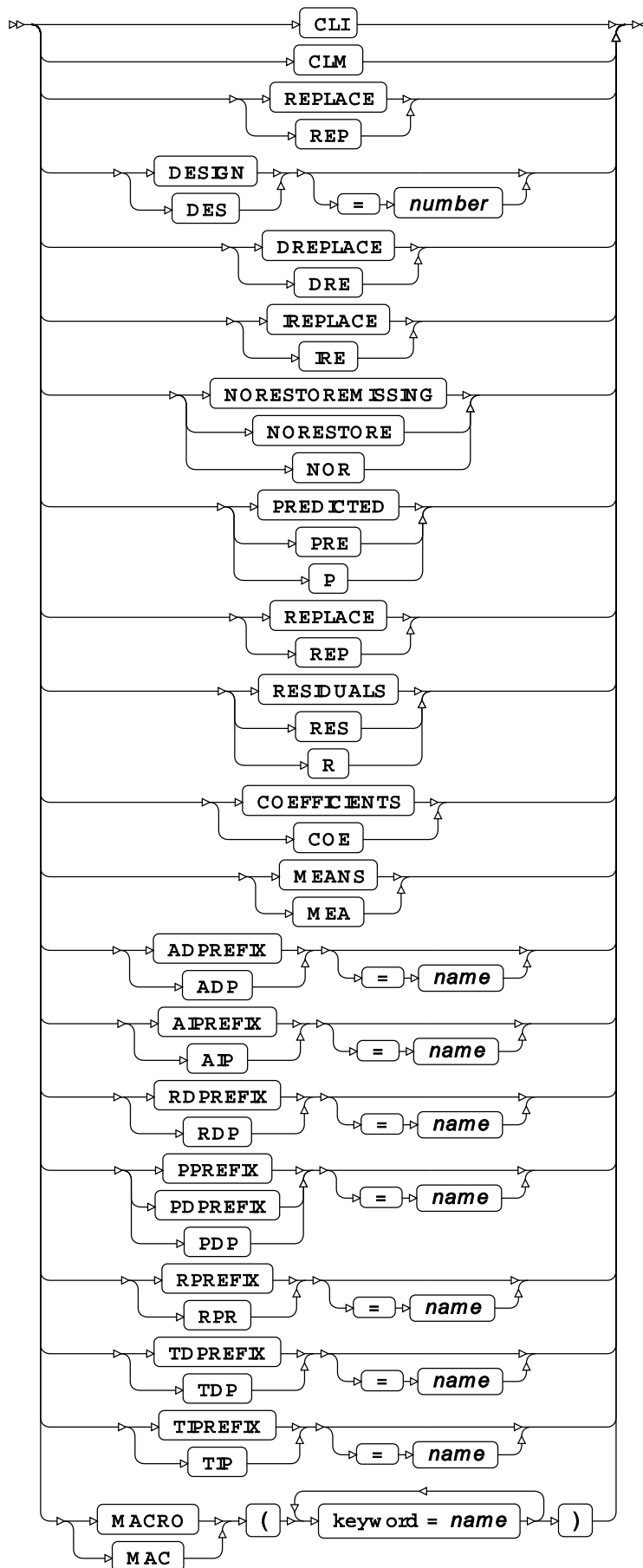


ⁱ See [Input dataset](#) (page 16).

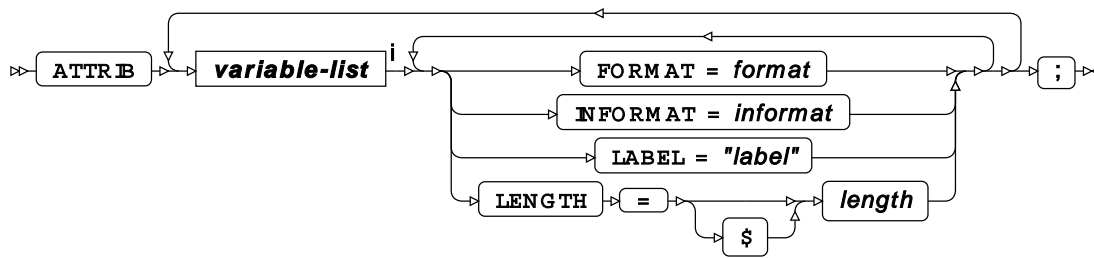
a-options



o-options

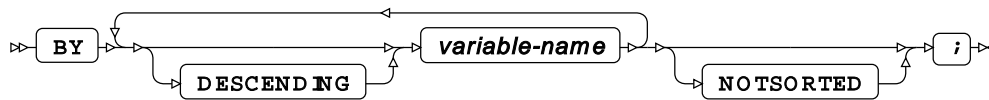


ATTRIB

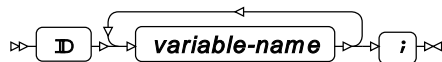


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



ID



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

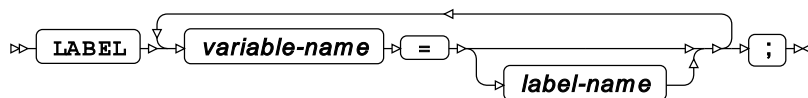


INFORMAT

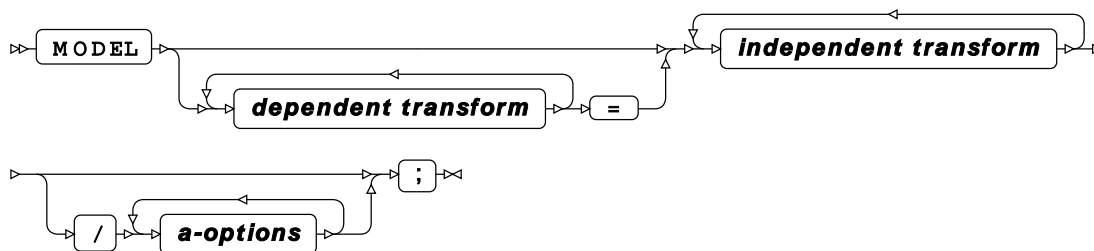


ⁱ See *Variable Lists* [↗](#) (page 32).

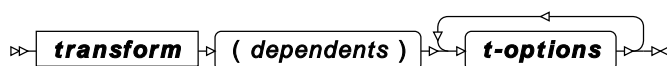
LABEL



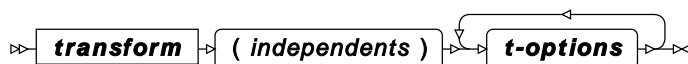
MODEL

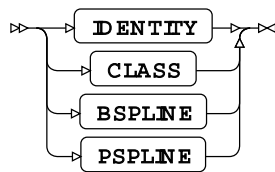


dependent transform

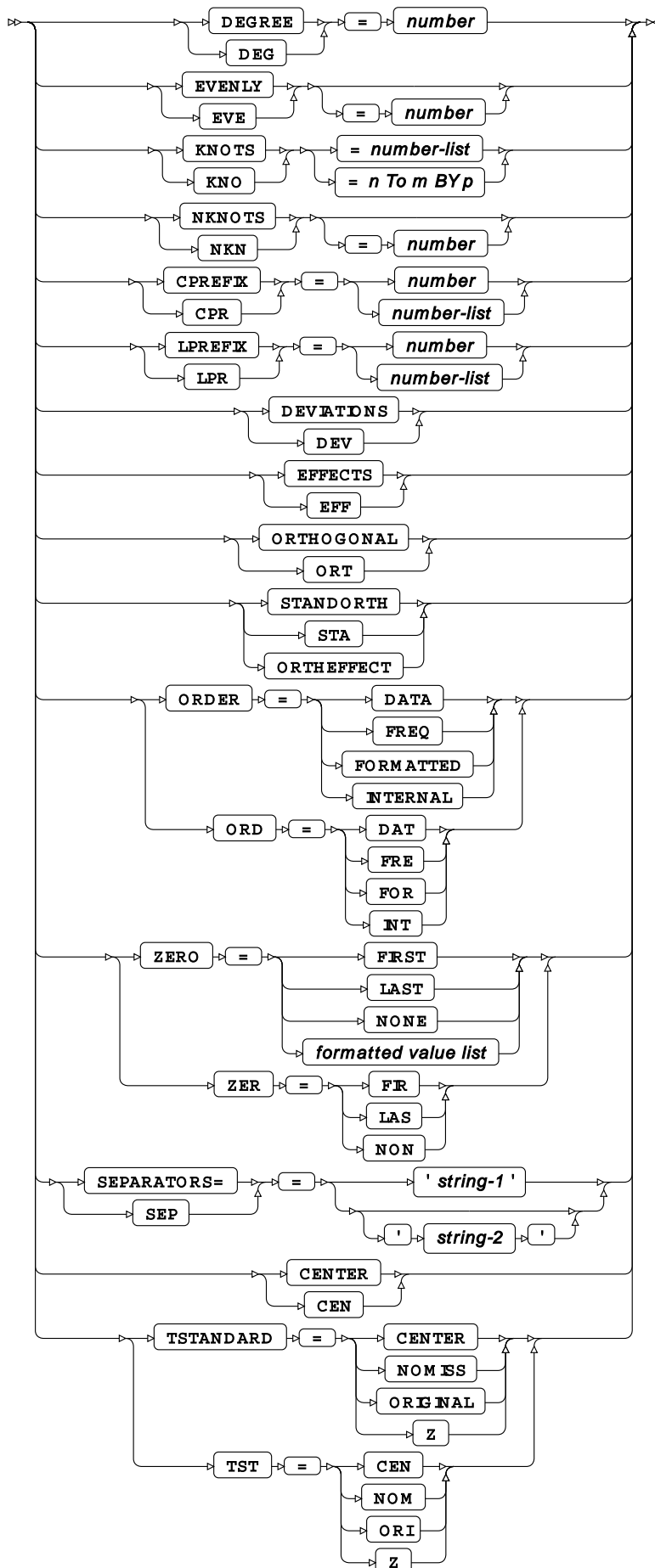


independent transform

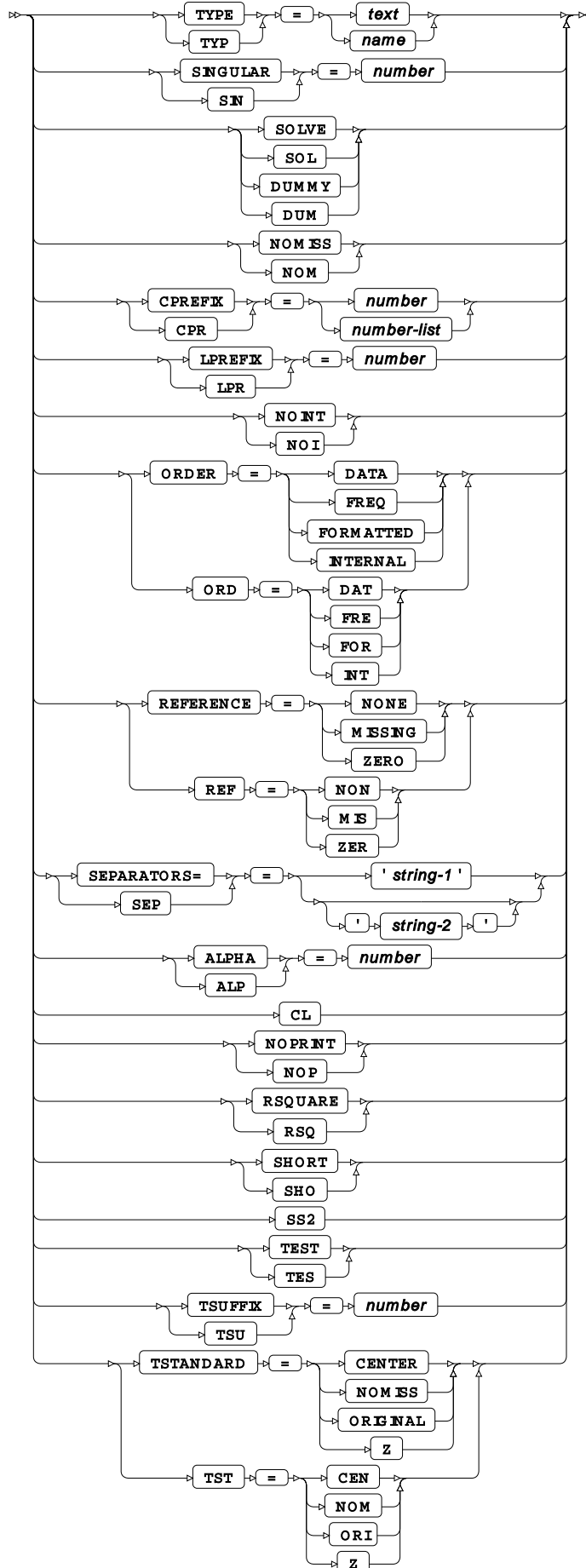


transform

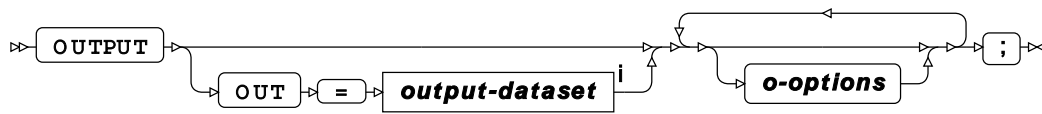
t-options



a-options

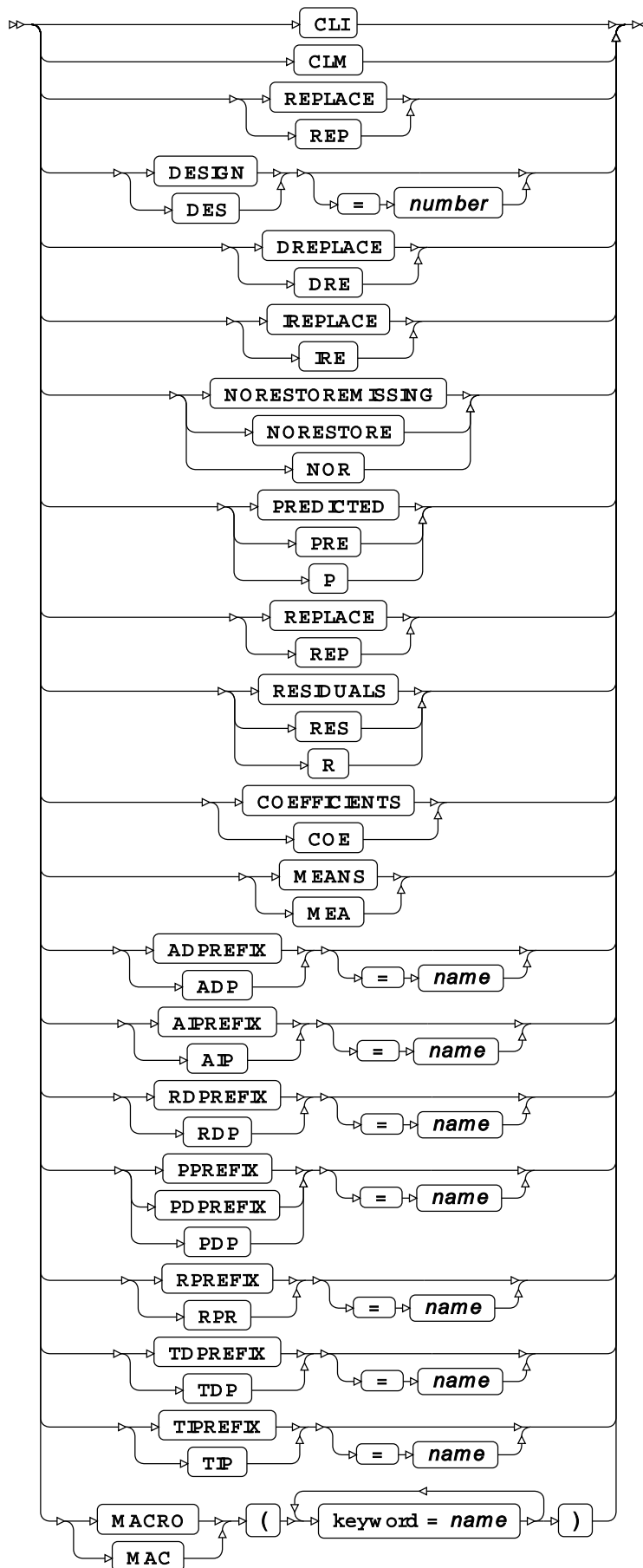


OUTPUT



ⁱ See *Output dataset* [↗](#) (page 16).

o-options



WEIGHT



WHERE

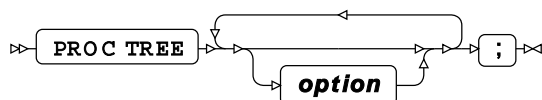


TREE procedure

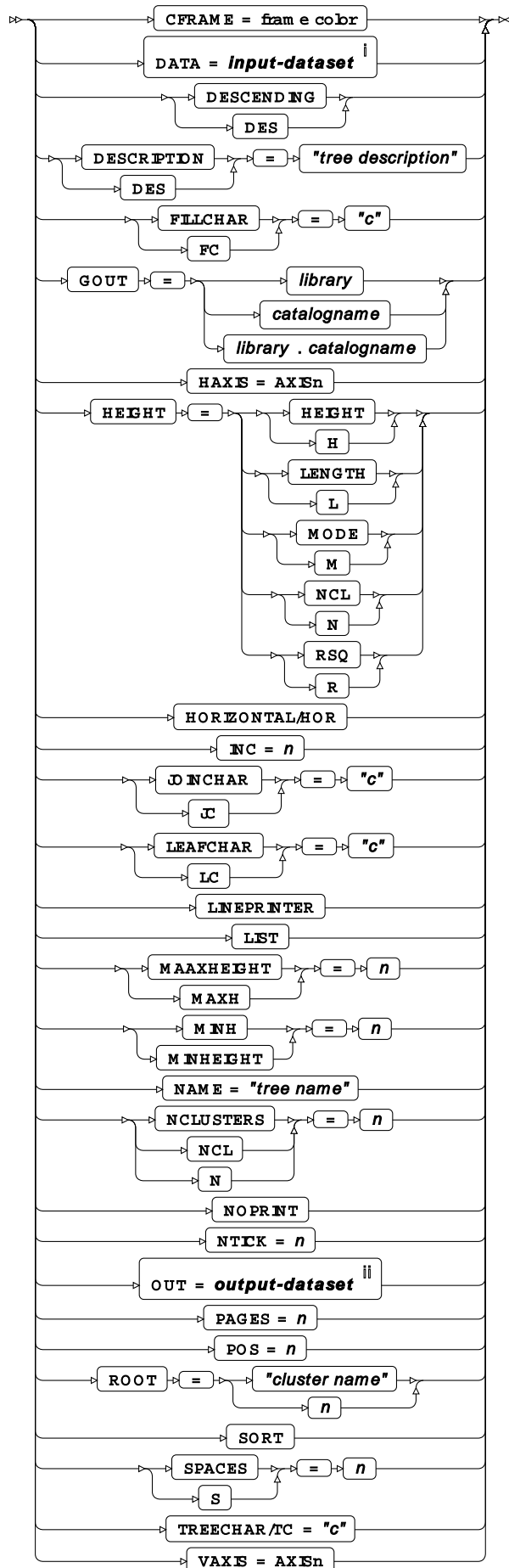
Supported statements

- *PROC TREE* [↗](#) (page 3153)
- *ATTRIB* [↗](#) (page 3155)
- *BY* [↗](#) (page 3155)
- *COPY* [↗](#) (page 3155)
- *FORMAT* [↗](#) (page 3155)
- *FREQ* [↗](#) (page 3156)
- *INFORMAT* [↗](#) (page 3156)
- *LABEL* [↗](#) (page 3156)
- *HEIGHT* [↗](#) (page 3156)
- *ID* [↗](#) (page 3156)
- *NAME* [↗](#) (page 3156)
- *PARENT* [↗](#) (page 3157)
- *WHERE* [↗](#) (page 3157)

PROC TREE



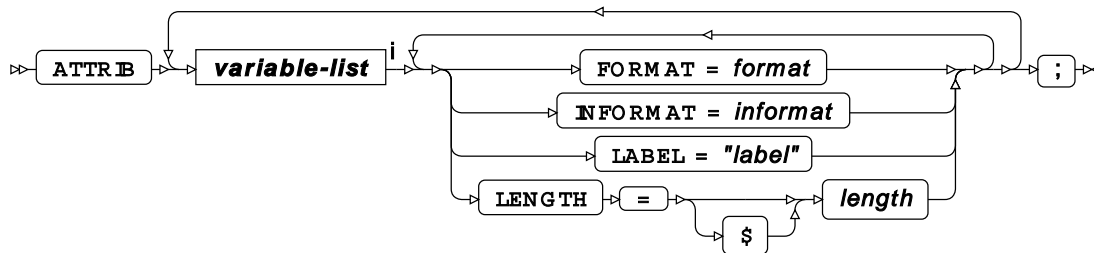
option



ⁱ See *Input dataset* [↗](#) (page 16).

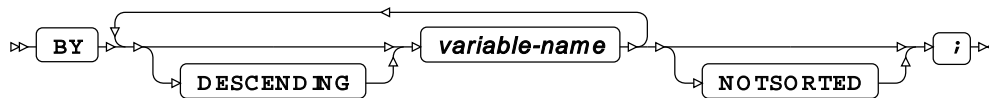
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

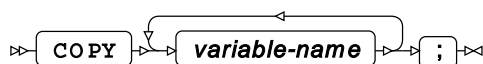


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



COPY



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

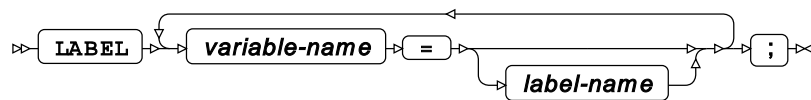


INFORMAT



ⁱ See [Variable Lists](#) (page 32).

LABEL



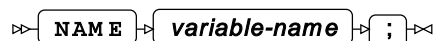
HEIGHT



ID



NAME



PARENT



WHERE

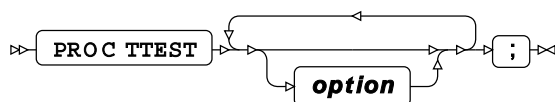


TTEST procedure

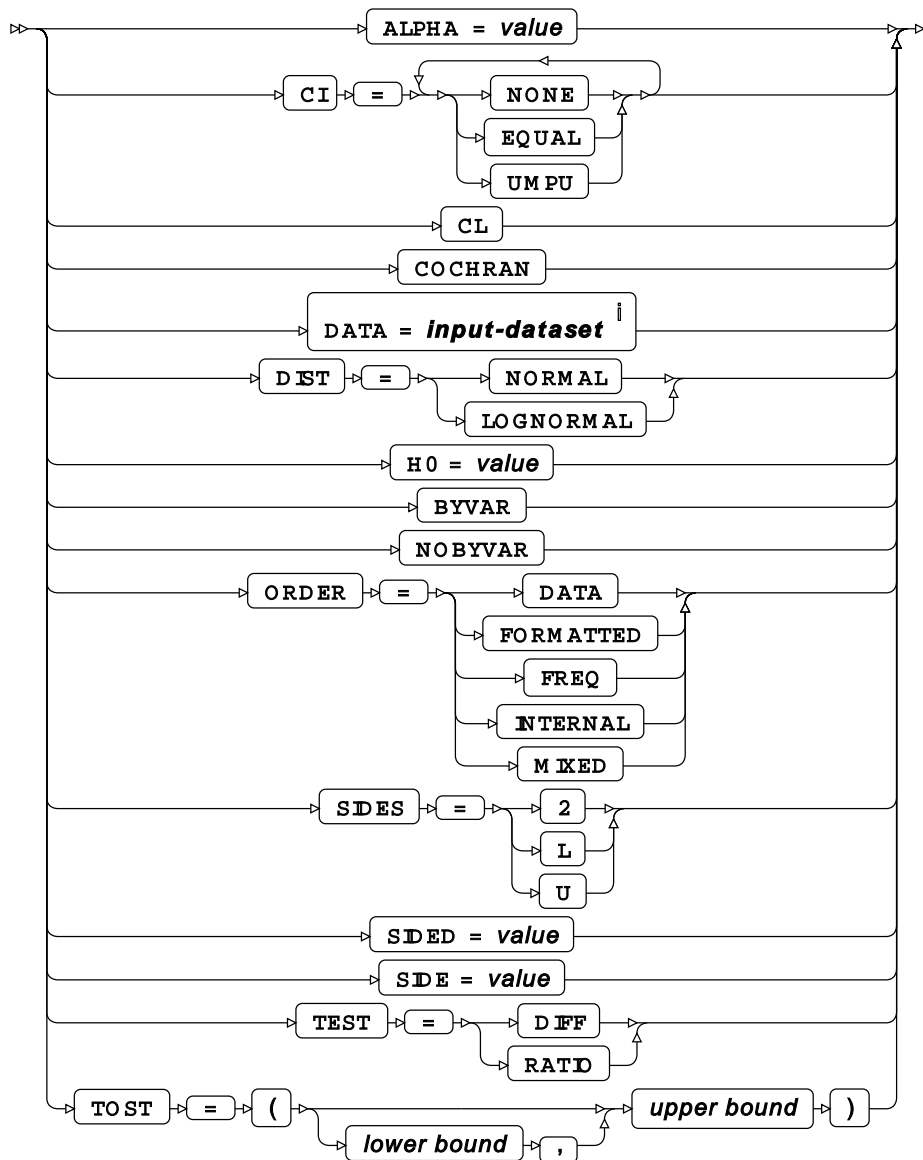
Supported statements

- *PROC TTEST* [↗](#) (page 3157)
- *ATTRIB* [↗](#) (page 3159)
- *BY* [↗](#) (page 3159)
- *CLASS* [↗](#) (page 3159)
- *FORMAT* [↗](#) (page 3159)
- *FREQ* [↗](#) (page 3159)
- *INFORMAT* [↗](#) (page 3160)
- *LABEL* [↗](#) (page 3160)
- *PAIRED* [↗](#) (page 3160)
- *VAR* [↗](#) (page 3161)
- *WEIGHT* [↗](#) (page 3161)
- *WHERE* [↗](#) (page 3161)

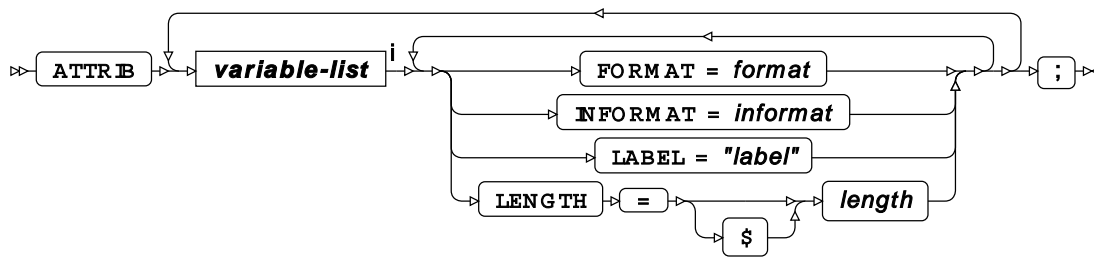
PROC TTEST



option

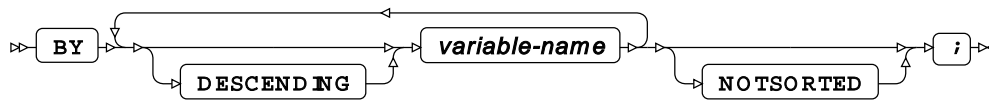
ⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

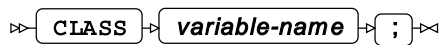


ⁱ See *Variable Lists* [↗](#) (page 32).

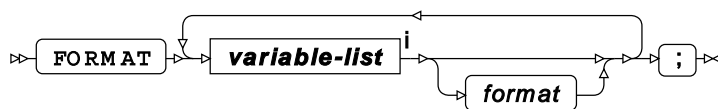
BY



CLASS



FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

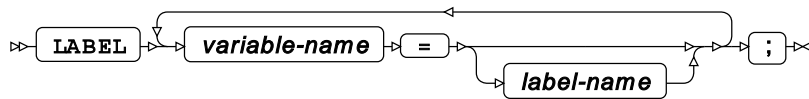


INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

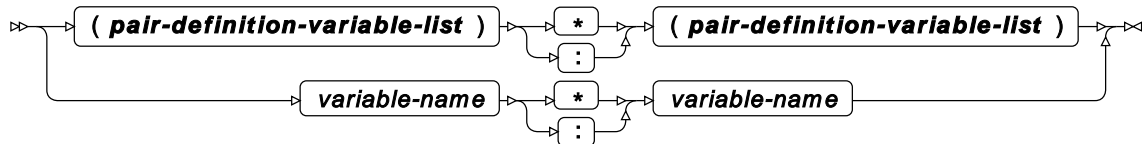
LABEL



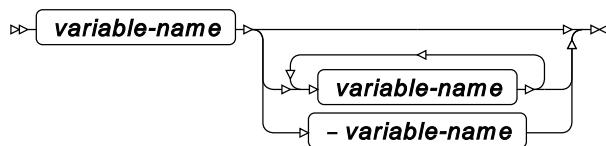
PAIRED



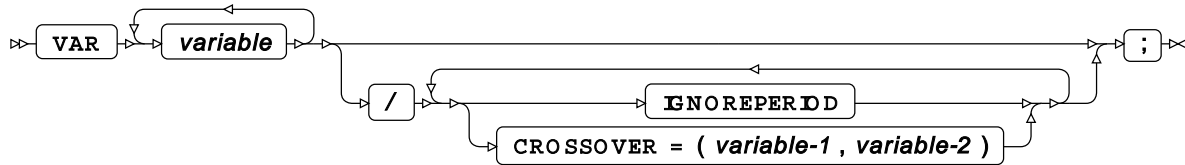
pair-definition



pair-definition-variable-list



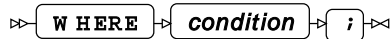
VAR



WEIGHT



WHERE

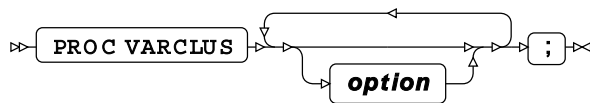


VARCLUS procedure

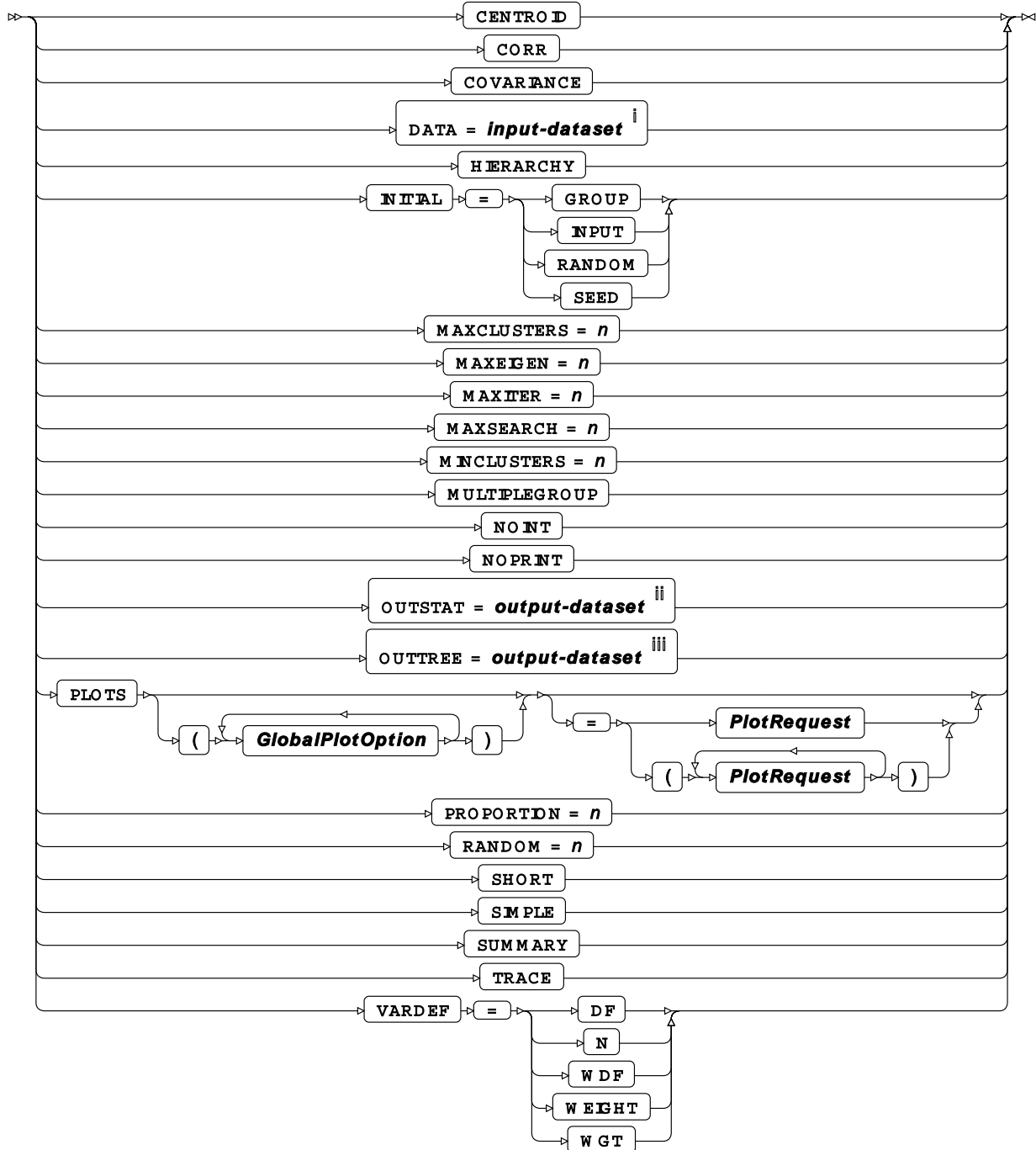
Supported statements

- *PROC VARCLUS* [↗](#) (page 3162)
- *ATTRIB* [↗](#) (page 3164)
- *BY* [↗](#) (page 3164)
- *FORMAT* [↗](#) (page 3164)
- *FREQ* [↗](#) (page 3164)
- *INFORMAT* [↗](#) (page 3164)
- *LABEL* [↗](#) (page 3165)
- *PARTIAL* [↗](#) (page 3165)
- *SEED* [↗](#) (page 3165)
- *VAR* [↗](#) (page 3165)
- *WEIGHT* [↗](#) (page 3165)
- *WHERE* [↗](#) (page 3165)

PROC VARCLUS



option

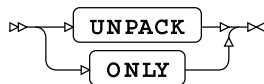


ⁱ See *Input dataset* [↗](#) (page 16).

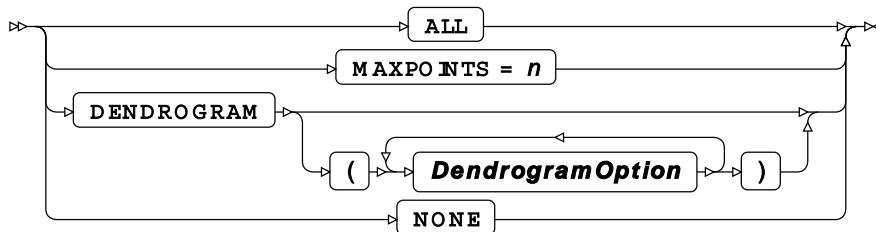
ⁱⁱ See *Input dataset* [↗](#) (page 16).

ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

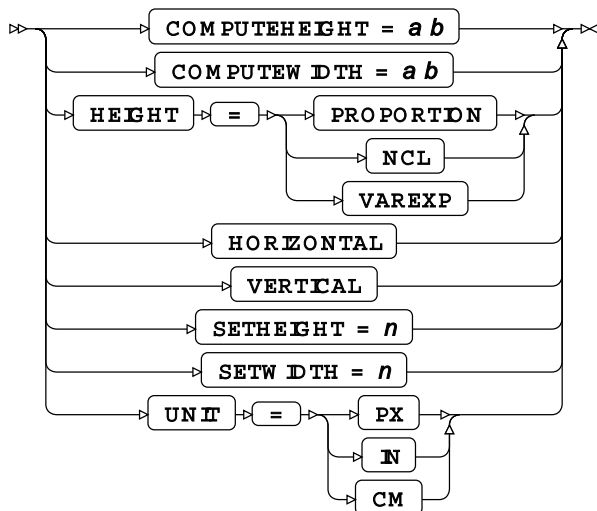
GlobalPlotOption



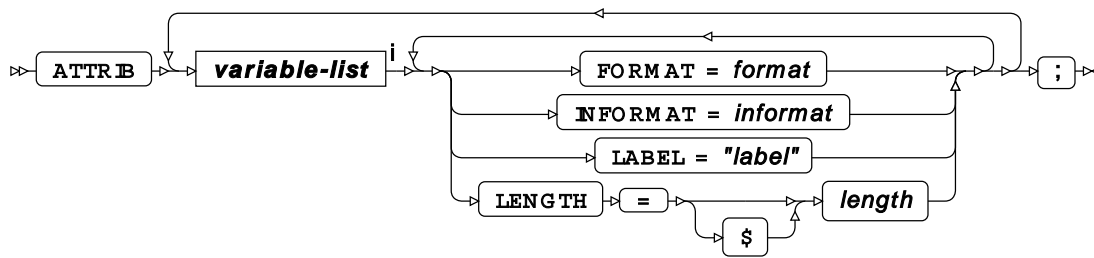
PlotRequest



DendrogramOption

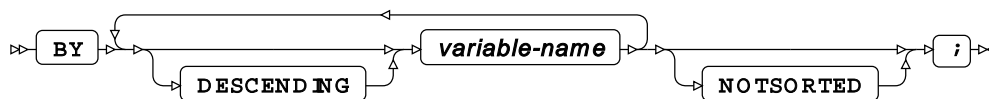


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

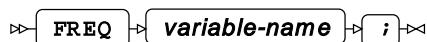


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

FREQ

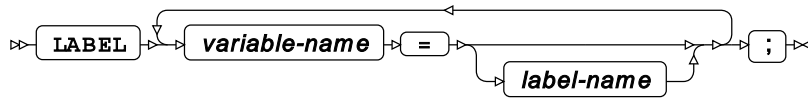


INFORMAT

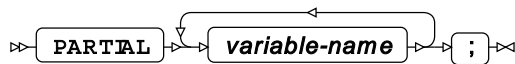


ⁱ See *Variable Lists* [↗](#) (page 32).

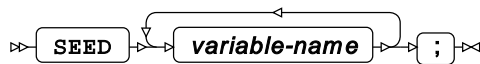
LABEL



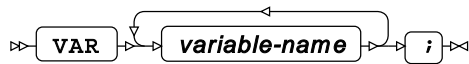
PARTIAL



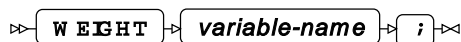
SEED



VAR



WEIGHT



WHERE

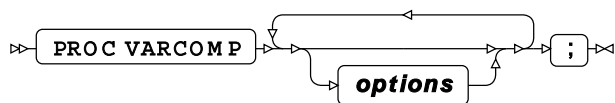


VARCOMP procedure

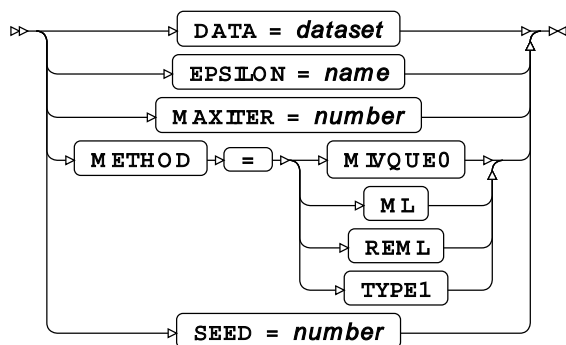
Supported statements

- *PROC VARCOMP* [↗](#) (page 3166)
- *ATTRIB* [↗](#) (page 3167)
- *BY* [↗](#) (page 3167)
- *CLASS* [↗](#) (page 3167)
- *FORMAT* [↗](#) (page 3167)
- *INFORMAT* [↗](#) (page 3167)
- *LABEL* [↗](#) (page 3168)
- *MODEL* [↗](#) (page 3168)
- *WHERE* [↗](#) (page 3168)

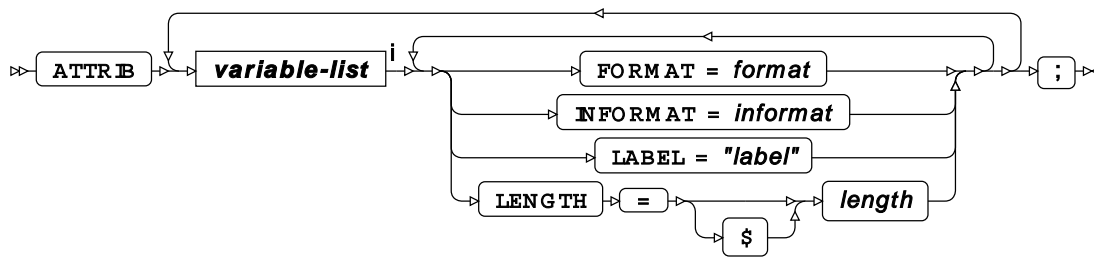
PROC VARCOMP



options

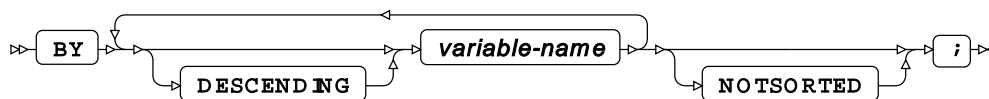


ATTRIB

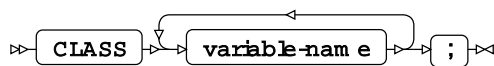


ⁱ See *Variable Lists* [↗](#) (page 32).

BY



CLASS



FORMAT



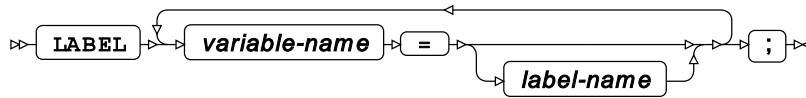
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

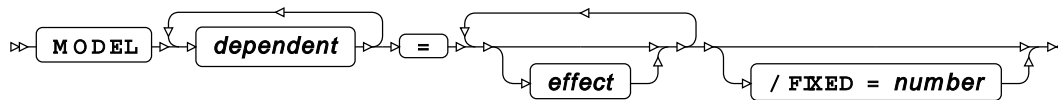


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



MODEL



WHERE



WPS Machine Learning

The WPS machine learning procedures provide access to a number of powerful data analysis and data modelling algorithms such as classification and regression trees, mixture models and support vector machines.

DECISIONFOREST procedure ↗	3170
The DECISIONFOREST procedure enables you to build a decision forest from an input dataset. You can then use the decision forest model to analyse other datasets.	
DECISIONTREE procedure ↗	3188
The DECISIONTREE procedure enables you to build a decision tree from an input dataset. You can then use the decision tree model to analyse other datasets.	
GMM procedure ↗	3209
The GMM procedure enables you to build a Gaussian mixture model (GMM) from an input dataset. You can then use the GMM to analyse other datasets.	
MLP procedure ↗	3250
The MLP procedure enables you to build a multilayer perceptron (MLP) from an input dataset. You can then use the MLP model to analyse other datasets.	
OPTIMALBIN procedure ↗	3296
The OPTIMALBIN procedure enables you to perform optimal binning on an input dataset.	
SEGMENT procedure ↗	3314
The SEGMENT procedure enables you to segment an input dataset to highlight similarities and differences in the data.	
SVM procedure ↗	3349
The SVM procedure enables you to build a support vector machine (SVM) from an input dataset. You can then use the SVM model to analyse other datasets.	

DECISIONFOREST procedure

The DECISIONFOREST procedure enables you to build a decision forest from an input dataset. You can then use the decision forest model to analyse other datasets.

About decision forests

A decision forest is a prediction model which uses a collection of *decision trees* to predict the value of a *target* (or response) variable from the values of one or more *input* (or predictor) variables.

For more information about decision trees see *About decision trees* [↗](#) (page 3188).

The target variable may be discrete, in which case the decision forest is a *classification* forest, or it may be continuous, in which case the decision forest is a *regression* forest.

Each decision tree in the decision forest is generated using a random selection of observations from the training dataset. Each node in each tree is split using a random selection of predictor variables. Then each observation is run through each tree to determine the value of the target variable predicted by that tree. For each observation, the value predicted by the decision forest is a function of the predicted values from the decision trees in the collection. The two separate sources of randomness help to ensure that the final model does not over fit the data.

The algorithms for building classification and regression forests are based on algorithms developed by Leo Breiman and Adele Cutler, described in:

- Breiman, L., 2001, Random Forests. In: *Machine Learning*, Volume 45 Issue 1, October 1 2001, pp 5-32 and available online at <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf> [↗](#) (accessed 30th May 2018)
- <https://www.stat.berkeley.edu/~breiman/RandomForests> [↗](#) (accessed 30th May 2018).

Building the decision trees

Each tree in the decision forest is built using the CART algorithm developed by Breiman et. al. (Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984). The *Gini Impurity* criterion is used to determine the best variable to use to split the observations at each node of the tree.

To build each decision tree, N observations are chosen randomly from the training dataset, where N is the total number of observations in the training dataset. The observations are sampled with replacement, so for each tree, some observations will be used more than once, and some observations will not be used at all. The observations that are not used to build a specific tree are referred to as the out-of-bag (OOB) observations for that tree, and are used later to derive the *OOB error estimate*, an estimate of the prediction error of the random forest.

At each node in each tree, a subset of the input variables is randomly selected from the specified input variables. For each of these variables, the split which gives the largest decrease in Gini Impurity is calculated. The variable which gives the largest overall decrease in Gini Impurity is selected as the variable to use for the split.

Stopping criteria

When any one of the following occurs, a node is not split further:

- the node is pure, that is, the target value is the same for all the observations in the node
- the number of observations in the node is less than or equal to the specified minimum split size (specified using either the `MINSPLITSIZE` option or the `MINSPLITSIZERATIO` option)
- the maximum decrease in Gini Impurity (for any of the randomly selected variables that were considered for the split) is less than the specified minimum improvement (specified using the `MINIMPROVEMENT` option).

Predicted values

When a tree has been completed, it can process new data observations to derive predicted values for the target variable. For each observation, the predictions from each tree are combined to derive the overall prediction from the forest.

For a regression forest, each tree predicts a single value for the target variable. The overall prediction is simply the mean of the predictions from the individual trees.

For a classification forest, each tree predicts the probability associated with each possible class value of the target variable. There are two ways that the predictions from each tree in the forest can be combined to produce the overall prediction.

- If `CLASSCOMB=MEANPROB` is specified, the overall predicted probability for a class is the mean of the probabilities predicted by each tree for that class.
- If `CLASSCOMB=VOTE` (the default) is specified, each tree casts a vote for a single class. Each tree votes for the class for which it has calculated the highest predicted probability. If the predicted probabilities for two or more classes are equal, a random selection is made. The overall predicted probability for a class is the proportion of trees in the forest that voted for that class.

OOB error estimate

The OOB error estimate is calculated from the OOB observations for each tree (the observations that were not randomly selected to build the tree).

After each tree has been built, each OOB observation for that tree is run through the tree to derive the predicted value for that observation. When the entire forest of trees has been completed, each observation that was OOB for one or more trees is considered. For each observation, the predicted values from the trees where the observation was OOB are combined to produce the overall OOB

predicted value for that observation. As above, for regression trees, the overall predicted value is the mean of the predictions from the individual trees. For classification trees, the predicted value for each class is either the mean of the probabilities predicted by the individual trees for that class, or the proportion of trees that voted for that class, depending on the `CLASSCOMB` option selected.

The OOB error estimate is then calculated as the percentage of OOB predicted values that are incorrect (for classification trees), or the mean squared error (for regression trees). Since the OOB predicted values are derived from observations that were not used to train the trees that predicted the values, the OOB error estimate is likely to be a good indicator of the overall error when the decision forest is used to score another dataset.

Input variable selection

The number of input variables that are randomly selected at each node is configurable using the `NUMSPLITINPUTS` option of the `PROC DECISIONFOREST` statement.

The optimal value for `NUMSPLITINPUTS` is likely to be significantly less than the total number of input variables. Lower values tend to reduce the *correlation* between the trees in the forest (a measure of how similar are the predictions made by each tree). Lower values also reduce the *strength* of the trees in the forest (a measure of how accurate each tree is at predicting the target variable). A forest with a low correlation between trees is desirable, but a forest with a low strength is undesirable. Higher values for `NUMSPLITINPUTS` tend to increase both the correlation (undesirable) and the strength (desirable).

To choose a suitable value for `NUMSPLITINPUTS`, you can run `PROC DECISIONFOREST` with different values of `NUMSPLITINPUTS`, then use the OOB error estimate measure from each run to determine the optimal value.

Missing data

Observations with missing data can be ignored using the `EXCLUDEMISS` option of the `PROC DECISIONFOREST` statement. If `EXCLUDEMISS` is not specified, then missing data is handled as follows:

- for categorical (discrete) variables, missing values are replaced with the mode of the non-missing values for that variable
- for non-categorical (continuous) variables, missing values are replaced with the median of the non-missing values for that variable.

Using the DECISIONFOREST procedure

This example shows how to use `PROC DECISIONFOREST` to build a decision forest from an input dataset and to measure how well the decision forest can predict the target variable.

Dataset

This example uses the Titanic dataset. The Titanic dataset consists of data about the passengers on the Titanic, and whether they survived or not. The data has been cleaned to replace some missing values, then split into a training dataset, `titanic_training.wpd`, containing two-thirds of the observations and a test dataset, `titanic_test.wpd`, containing one-third of the observations. The Titanic dataset is courtesy of Encyclopedia Titanica (encyclopedia-titanica.org [↗](#)). Used with permission.

Code example

In this example, a classification decision forest is built using the input variables `class`, `sex`, `age`, `num_sib_spouse` (the number of siblings and spouses the passenger was travelling with) and `num_parent_child` (the number of parents or children the passenger was travelling with) as predictors of the target variable, `survived`. The algorithm builds a decision forest model of 600 trees from the training dataset (`NUMTREES=600`).

The `OUTPUT` statement specifies that an output dataset is produced containing the input observations, and the predicted value of the target variable for each observation. The `PROBABILITY` option specifies that the output dataset includes an additional variable for each possible category that the target variable can take. For each observation, these variables give the probability that the actual target value is in that category.

The `SCORE` statement specifies that the decision forest model is used to score the test dataset. Again, the `PROBABILITY` option specifies that the scored dataset includes an additional variable for each possible category that the target variable can take, giving the probability that the actual value is in that category.

In this example, the test dataset and the training dataset are both located in a library called `TESTDATA`.

```
PROC DECISIONFOREST
  DATA = TESTDATA.titanic_training
  NUMTREES=600;
INPUT class/LEVEL=ORDINAL;
INPUT sex/LEVEL=NOMINAL;
INPUT age/LEVEL=INTERVAL;
INPUT num_sib_spouse/LEVEL=ORDINAL;
INPUT num_parent_children/LEVEL=ORDINAL;
TARGET survived/LEVEL=BINARY;
OUTPUT out=titanic_training_scored PROBABILITY;
SCORE DATA=TESTDATA.titanic_test OUT=DF_titanic_test_scored PROBABILITY;
RUN;
```

Target Summary

The Target Summary table shows the target variable that the decision forest is required to predict. In this case, the target variable is discrete with binary (yes or no) values.

Target Summary Target variable: survived					
Name	Discrete or continuous?	Level type	Number of categories	Order type	Descending?
survived	DISCRETE	BINARY	2	INTERNAL	NO

Input Summary

The Input Summary table summarises the input variables, their types, categories and the other properties.

Input Summary Target variable: survived					
Name	Discrete or continuous?	Level type	Number of categories	Order type	Descending?
SEX	DISCRETE	NOMINAL	2	INTERNAL	NO
CLASS	DISCRETE	ORDINAL	3	INTERNAL	NO
NUM_SIB_SPOUSE	DISCRETE	ORDINAL	7	INTERNAL	NO
NUM_PARENT_CHILD	DISCRETE	ORDINAL	8	INTERNAL	NO
AGE	CONTINUOUS	INTERVAL	.	INTERNAL	NO

Run Summary

The Run Summary table summarises information about the run, for example, the input dataset, the decision forest parameters that were used, and the number of data items processed.

Run Summary Target variable: survived	
Attribute	Value
Input dataset	TESTDATA.titanic_training
Exclude missings?	NO
Number of trees	600
Number of inputs per split	2
Minimum improvement	0.00000
Number of cases	873
Min split size for run	1

OOB Classification Performance

The OOB Classification Performance table shows the out-of-bag (OOB) error estimate from the decision forest. This is the proportion of observations in the training dataset that were incorrectly classified using OOB classification.

To calculate the predicted OOB classification for an observation, the observation is classified using each tree for which the observation was out-of-bag (that is, each tree for which the observation wasn't selected to train that tree). The most popular value is chosen as the predicted OOB classification for that observation. The OOB error estimate is the proportion of observations which are incorrectly classified using this algorithm.

In this example, the OOB error estimate is 21.08%, which means that 21.08% of the predicted category values for the OOB observations are incorrect for this dataset.

OOB Classification Performance	
Target variable: survived	
OOB Performance Summary	
Out-Of-Bag Error Estimate (%)	
	21.08

OOB Classification Table

The OOB Classification table shows the *confusion matrix* for the OOB classifications (the predicted values for each observation, as predicted by only the trees for which the observation was out-of-bag).

The rows represent the actual category values, and the columns represent the category values predicted by the OOB classification. The table entries show how many observations of each category value were mapped to each of the predicted category values, so the leading diagonal contains the numbers of observations that were classified correctly, and the other entries in the table contain the observations that were misclassified.

In this example, 468 non-survivors (actual category 'no') were correctly classified, but 61 non-survivors were incorrectly classified as survivors. 221 survivors (actual category 'yes') were correctly classified, but 123 survivors were incorrectly classified as non-survivors.

OOB Classification Table		
Target variable: survived		
Predicted Category Frequencies		
Actual Category	no	yes
no	468	61
yes	123	221

Classification Table

The Classification table shows the *confusion matrix* for the overall predictions from the decision forest. As before, the rows represent the actual category values, and the columns represent the category values predicted by the decision forest model. The leading diagonal contains the numbers of observations that were classified correctly, and the other entries in the table contain the observations that were misclassified. In this case, the predicted value for each observation is derived using all the trees in the decision forest, regardless of whether the observation was used to train the tree, or was out-of-bag for that tree.

In this example, 487 non-survivors (actual category 'no') were correctly classified, but 42 non-survivors were incorrectly classified as survivors. 244 survivors (actual category 'yes') were correctly classified, but 100 survivors were incorrectly classified as non-survivors.

Classification Table		
Source: TESTDATA.titanic_training		
Predicted value: survived		
Predicted Category Frequencies		
Actual Category	no	yes
no	487	42
yes	100	244

Classification Performance

The Classification Performance table shows the error estimate for the final predictions from the decision forest. This is the proportion of observations in the training dataset that were incorrectly classified by the decision forest model.

In this example, the error estimate is 16.27%, which means that 16.27% of the predicted category values are incorrect for the observations in this dataset. This error estimate is lower than the OOB error estimate of 21.08%, which is to be expected, since this value includes predictions for observations that were used to train the forest.

Classification Performance	
Source: TESTDATA.titanic_training	
Predicted value: survived	
Performance Summary	
Error Estimate (%)	
16.27	

Score Classification Table

The Score Classification table shows the confusion matrix for the test dataset that the decision forest model is being used to score. As before, the rows represent the actual category values, and the columns represent the category values predicted by the decision forest model. The leading diagonal contains the numbers of observations that were classified correctly, and the other entries in the table contain the observations that were misclassified.

In this example, 249 non-survivors (actual category 'no') were correctly classified, but 31 non-survivors were incorrectly classified as survivors. 108 survivors (actual category 'yes') were correctly classified, but 48 survivors were incorrectly classified as non-survivors.

```
Score Classification Table
Source: TESTDATA.titanic_test
Predicted value: survived
```

Predicted Category Frequencies		
Actual Category	no	yes
no	249	31
yes	48	108

Note that if the dataset to be scored does not contain actual category values for comparison, the Score Classification table is still produced, but it is empty.

Score Classification Performance

The Score Classification Performance table shows the error estimate for the test dataset that the decision forest model is being used to score. This is the proportion of observations in the dataset being scored that were incorrectly classified by the decision forest model.

In this example, the error estimate is 18.12%, which means that 18.12% of the predicted category values are incorrect for the dataset being scored.

```
Score Classification Performance
Source: TESTDATA.titanic_test
Predicted value: survived
```

Score Performance Summary
Score Error Estimate (%)
18.12

Note that if the dataset to be scored does not contain actual category values for comparison, the Score Classification Performance table is still produced, but it is empty.

Changing the number of variables chosen for each split

The example above uses the default value of `NUMSPLITINPUTS`, which in this case is 2.

If the example is changed to set `NUMSPLITINPUTS=3`, the OOB error estimate increases slightly from 21.08% to 22.45% but the error estimate for the classification performance decreases from 16.27% to 12.94%. The score classification performance also increases, from 18.12% to 19.27%.

If the example is changed to set `NUMSPLITINPUTS=1`, the OOB error estimate again increases slightly from 21.08% to 22.45%. The error estimate for the classification performance also increases from 16.27% to 17.98% and the score classification performance also increases, from 18.12% to 19.27%.

This illustrates the effect of choosing the optimal number of variables to select randomly at each node. Setting `NUMSPLITINPUTS=3` creates a decision forest that fits the training data slightly better, but is slightly worse at scoring other similar datasets. Setting `NUMSPLITINPUTS=1` creates a decision forest that is a worse fit to the training data, and is also slightly worse at scoring other similar datasets. Overall, for this dataset, the default value of `NUMSPLITINPUTS=2` appears to give the best results.

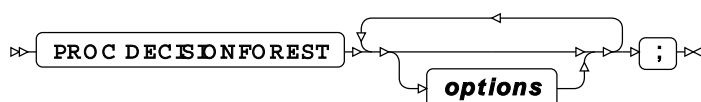
DECISIONFOREST procedure reference

Describes the syntax and options for `PROC DECISIONFOREST` and its contained statements.

<code>PROC DECISIONFOREST</code> ↗	3178
Specifies the algorithm and options to use to create a decision forest from the input dataset.	
<code>FREQ</code> ↗	3181
Specifies a variable containing the frequency associated with an observation.	
<code>INPUT</code> ↗	3181
Specifies the input (predictor) variables and the options to use for each.	
<code>OUTPUT</code> ↗	3183
Creates an output dataset containing the input observations and, for each, the value of the target variable as predicted by the decision forest model.	
<code>SCORE</code> ↗	3184
Uses the current decision forest model to score the data in the specified dataset.	
<code>TARGET</code> ↗	3185
Specifies the target (dependent) variable and any options that apply to the variable.	
<code>WEIGHT</code> ↗	3187
Specifies a variable in the input dataset giving the prior weight associated with each observation.	
<code>WHERE</code> ↗	3187
Restricts the observations to be processed.	

PROC DECISIONFOREST

Specifies the algorithm and options to use to create a decision forest from the input dataset.

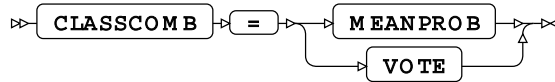


The `DECISIONFOREST` procedure is a machine learning algorithm that creates multiple tree-like decision models from the specified input dataset. Each node of each tree splits the data based on the values of a randomly-selected subset of the input variables. Each tree uses the CART algorithm and *Gini Impurity* to measure of the predictive power of the variables. The final predicted value assigned to the target variable is based on the aggregated predictions from all the trees in the forest.

Options

The following *options* are available:

CLASSCOMB



For a classification forest, specifies the way to combine the predicted probabilities from each tree in the forest to obtain the overall predicted probabilities from the forest.

This value is ignored for a regression forest.

The following values are available:

MEANPROB

The predicted probability for a class is the mean of the probabilities predicted by each tree for that class.

VOTE

The predicted probability for a class is the proportion of trees in the forest that voted for that class (that is, predicted that class to be the most likely class). If a tree predicted that two or more classes were equally likely, a random choice is made between these classes.

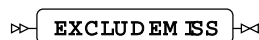
This is the default value.

DATA



Specifies the training dataset used to construct the decision forest. If neither the `DATA` nor the `INMODEL` options are specified, the most recently-created dataset is used as the training dataset. The `DATA` option cannot be specified if the `INMODEL` option is also specified.

EXCLUDEMISS



Specifies that observations with missing values are excluded when determining the best split at a node.

By default, observations with missing values are included. For classification forests, the missing value is replaced with the modal category for the target variable, and for regression forests, the missing value is replaced with the mean value of the target variable.

INMODEL



Specifies the location of a previously-saved, serialised decision forest model to be used to score another dataset. The dataset to be scored is specified using the `SCORE` statement.

The `INMODEL` option cannot be specified if the `DATA` option is also specified, or if any `INPUT` statements are included.

reference

The serialised model name is specified as *library-name.item-name*, where *library-name* is a library reference, and *item-name* is the item contained by the library. If *library-name* is not specified, the default `WORK` library is used.

MINIMPROVEMENT

⇒ `MINIMPROVEMENT` ⇒ `=` ⇒ *improvement* ⇒

Specifies the minimum improvement (decrease) in Gini Impurity required for the node to be split. The default `MINIMPROVEMENT` is 0.

MINSPLITSIZE

⇒ `MINSPLITSIZE` ⇒ `=` ⇒ *size* ⇒

Specifies the minimum number of observations a node must contain for the node to be split. If the node contains fewer observations than the specified minimum split size, the node is not split further. The default `MINSPLITSIZE` is 2 for trees in classification forests and 5 for trees in regression forests.

MINSPLITSIZERATIO

⇒ `MINSPLITSIZERATIO` ⇒ `=` ⇒ *ratio* ⇒

Specifies the minimum number of observations a node must contain for the node to be split, as a percentage of the observations in the dataset. If the node contains fewer observations than the specified minimum split size ratio, the node is not split further. The default value is 1 (percent).

NOPRINT

⇒ `NOPRINT` ⇒

Specifies that all ODS output is suppressed.

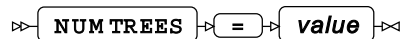
NUMSPLITINPUTS

⇒ `NUMSPLITINPUTS` ⇒ `=` ⇒ *value* ⇒

Specifies the number of input variables to randomly select each time a node is split. At each node in each tree in the decision forest, the algorithm selects this number of variables randomly from the total number of input variables and uses the values of the selected variables to determine the best split for that node.

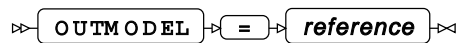
For a decision forest with N input variables, the default value is $\lfloor \sqrt{N} \rfloor$ (that is, $\text{floor}(\sqrt{N})$) for classification trees and $N/3$ for regression trees.

NUMTREES



Specifies the number of trees in the forest. The default number of trees is 500.

OUTMODEL



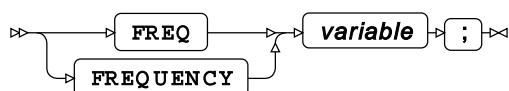
Specifies that the decision forest model created from the training dataset is saved as a serialised model in the specified location. This serialised model can later be used to score another dataset.

reference

The serialised model name is specified as *library-name.item-name*, where *library-name* is a library reference, and *item-name* is the item contained by the library. If *library-name* is not specified, the default `WORK` library is used.

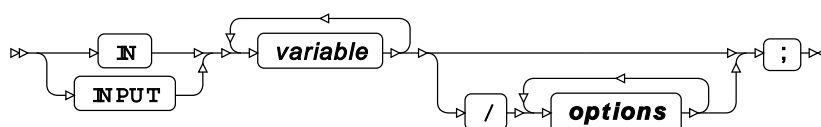
FREQ

Specifies a variable containing the frequency associated with an observation.



INPUT

Specifies the input (predictor) variables and the options to use for each.



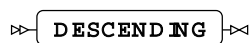
Together, the input variables specified in all the `INPUT` statements form the group of variables from which a random selection is made each time a node is split. If no `INPUT` statement is specified, all variables in the input dataset are regarded as input variables except for those which are specified the `TARGET` statement, and variables specified in the `FREQUENCY` and `WEIGHT` statements if used.

variable

A variable to which measures of predictive power are applied.

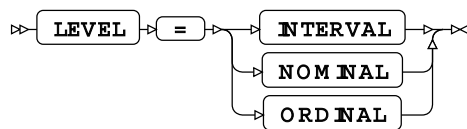
Options

The following *options* are available:

DESCENDING

Specifies a descending sort order for the variable. This option only applies if the value specified for the `ORDER` option is one of `FORMATTED`, `INTERNAL` or `UNFORMATTED`, as the sort order cannot be explicitly determined from these options. If `ORDER` has one of these values, and the `DESCENDING` option is not specified, the variable has an ascending sort order.

If the value specified for `ORDER` is not `FORMATTED`, `INTERNAL` or `UNFORMATTED` then the sort order for the variable is determined by the `ORDER` option.

LEVEL

Specifies the level for the input variables. The default `LEVEL` value is `INTERVAL`.

INTERVAL

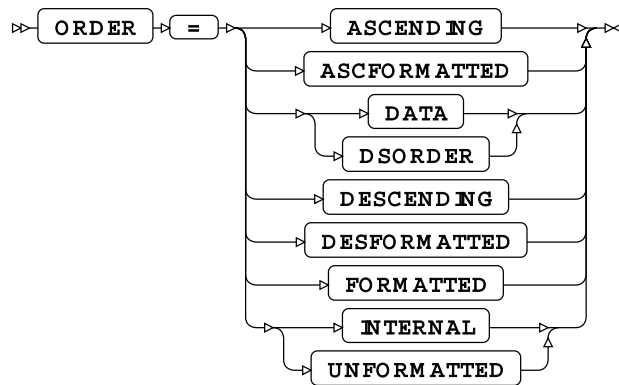
Specifies a continuous input variable with an implicit category ordering.

NOMINAL

Specifies a discrete input variable with no implicit ordering. When partitioning this variable into nodes in a decision tree, any category can be merged with any other category.

ORDINAL

Specifies a discrete input variable with an implicit category ordering. When partitioning this variable into nodes in a decision tree, only adjacent categories can be merged together.

ORDER

Specifies the order of the input variable. The default **ORDER** value is **INTERNAL**.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

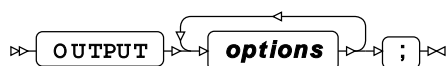
The variable is sorted by the formatted value. If the (separate) **DESCENDING** option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) **DESCENDING** option is also specified, the sort order is descending, otherwise, the sort order is ascending.

OUTPUT

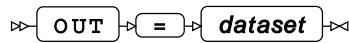
Creates an output dataset containing the input observations and, for each, the value of the target variable as predicted by the decision forest model.



Options

The following *options* are available:

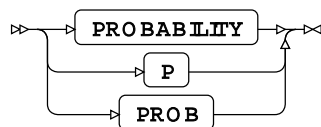
OUT



Specifies the name of the output dataset. The dataset contains the original data, and, for each observation, the predicted value for the target variable.

If **OUT** is not specified, the procedure creates the dataset as *DATA**n* in the **WORK** library, where *n* is incremented for each output dataset.

PROBABILITY



Specifies that, for classification decision forests, the output dataset includes a variable for each possible category that the target variable can take. For each observation, these variables give the probability that the actual target value is in that category.

This option is ignored if the forest is not a classification forest.

SCORE

Uses the current decision forest model to score the data in the specified dataset.



The **SCORE** statement takes the data in the specified dataset and scores it using the decision forest model defined in **PROC DECISIONFOREST**. The score results are saved in a table which can be printed or saved in an output dataset.

For each observation in the dataset being scored, the score results table shows the component that the observation is most likely to belong to, and the overall probability density of the distribution at that point. The score results table includes all the data in the input dataset, including observations with missing values. But observations with missing values for predictor variables are not scored.

Multiple **SCORE** statements can be specified if required.

Options

The following *options* are available:

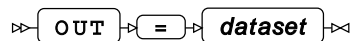
DATA

Specifies the dataset to score. All the predictor variables specified in the `INPUT` statement must be present in the dataset.

If `PROC DECISIONFOREST` includes a `BY` statement, the dataset to be scored must also contain all the variables mentioned in the `BY` statement, and must be sorted in the order of those variables.

The `DATA` option is mandatory.

OUT

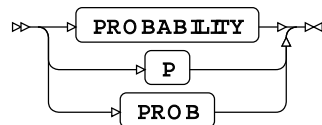


Specifies the dataset to which the score results are output. The dataset contains the original data, and, for each observation, the predicted value for the target variable.

If not specified, no output dataset is produced.

If `PROC DECISIONFOREST` specifies more than one output dataset for score results (for example, if there is more than one `SCORE` statement) then each output dataset must have a unique name.

PROBABILITY

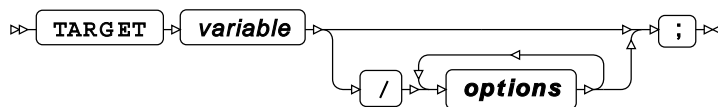


Specifies that, for classification decision forests, the output dataset includes a variable for each possible category that the target variable can take. For each observation, these variables give the probability that the actual target value is in that category.

This option is ignored if the forest is not a classification forest.

TARGET

Specifies the target (dependent) variable and any options that apply to the variable.



Only one `TARGET` statement is allowed in each `PROC DECISIONFOREST` statement. The `TARGET` statement must contain a single variable.

variable

The dependent variable.

Options

The following *options* are available:

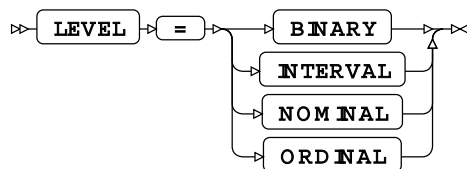
DESCENDING



Specifies a descending sort order for the variable. This option only applies if the value specified for the **ORDER** option is one of **FORMATTED**, **INTERNAL** or **UNFORMATTED**, as the sort order cannot be explicitly determined from these options. If **ORDER** has one of these values, and the **DESCENDING** option is not specified, the variable has an ascending sort order.

If the value specified for **ORDER** is not **FORMATTED**, **INTERNAL** or **UNFORMATTED** then the sort order for the variable is determined by the **ORDER** option.

LEVEL



Specifies the level for the target variable. The default **LEVEL** value is **NOMINAL**.

INTERVAL

Specifies a continuous target variable containing an implicit category ordering.

NOMINAL

Specifies a discrete target variable with no implicit ordering.

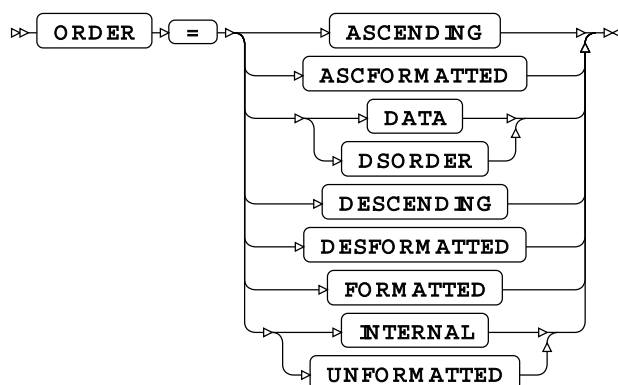
ORDINAL

Specifies a discrete target variable with an implicit category ordering.

BINARY

Specifies a target variable that can take one of two values.

ORDER



Specifies the order of target variable. The default `ORDER` value is `INTERNAL`.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

The variable is sorted by the formatted value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

WEIGHT

Specifies a variable in the input dataset giving the prior weight associated with each observation.

➤ `WEIGHT` ➤ `variable` ➤ `;` ➤

WHERE

Restricts the observations to be processed.

➤ `WHERE` ➤ `condition` ➤ `;` ➤

DECISIONTREE procedure

The DECISIONTREE procedure enables you to build a decision tree from an input dataset. You can then use the decision tree model to analyse other datasets.

About decision trees

A decision tree is a statistical model organised as a tree structure. It uses a succession of logic rules to predict the value of a *target* (or response) variable from the values of one or more *input* (or predictor) variables.

The decision tree is built from a training dataset where the values of the target variable are known. The decision tree can then be used to predict the values of the target variable in other datasets where the values of the target variable may not be known.

If the target variable is discrete, the tree is a *classification* tree. If the target variable is continuous, the tree is a *regression* tree.

A decision tree consists of a series of *nodes*, each containing observations from the dataset. The top-level node is the root node and contains all the observations. At each node, the tree-building algorithm uses the observations at that node to determine the best input variable to use to split the tree further. The input variable that gives the greatest increase in information, according to some specified criterion, is used to split that node of the tree into two or more *branches* (or *partitions*). For a continuous input variable, the value of the variable is compared against a threshold value. For a discrete input variable, the value of the variable is compared against two or more subsets of possible values.

Each node can be partitioned into lower level nodes. As the depth of the tree increases, each node contains fewer and fewer observations. Tree growth stops when some specified stopping criteria is reached. For example, you could specify that growth stops when a further split would cause the number of observations in a node to drop below a specified minimum, or when the tree reaches a specified maximum node depth.

Another important concept of decision trees is *pruning*. When a decision tree is built from a training dataset, the tree may *over-fit* that data and hence perform poorly against other test data. Pruning allows you to build a tree which over-fits, and then to remove any lower nodes which are statistically insignificant according to some specified test. This increases the likelihood that the decision tree will perform well against a new dataset. The pruning options available and the way that pruning is carried out depend upon the algorithm used to build the tree.

Tree-building algorithms

The PROC DECISIONTREE METHOD option specifies the tree building algorithm to use. PROC DECISIONTREE supports the following algorithms:

- **METHOD=BRT** uses the Binary Response Tree (BRT) algorithm developed by WPL for predicting binary target variables. The mechanism for creating a split uses algorithms developed by Raymond Anderson for partitioning the values of an input variable (Anderson, R. *The Credit Scoring Toolkit*, Oxford Press, 2007) and measures of predictive power developed by Mamdouh Reefat (Reefat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*, Lulu.com, 2016).

This algorithm applies to classification trees with binary target variables.

- **METHOD=C4.5** uses the C4.5 algorithm developed by Ross Quinlan (Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993).

This algorithm applies to any classification trees.

- **METHOD=CART** uses the Classification and Regression Trees (CART) algorithm developed by Breiman et. al. (Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984).

This algorithm applies to any classification or regression tree.

Node-splitting criteria

Use the `PROC DECISIONTREE CRITERION` option to specify the criterion to use to determine the best split at each node of the tree. The available options depend on the `METHOD` option chosen.

METHOD=BRT

When `METHOD=BRT`, the `CRITERION` options are measures of predictive power. The following `CRITERION` options are available:

- `CRITERION=CHISQUARED` uses *Pearson's Chi-Squared* statistic
- `CRITERION=ENTROPYVAR` uses *Entropy Variance*
- `CRITERION=GINIVARIANCE` uses *Gini Variance*
- `CRITERION=INFORMATIONVALUE` uses *Information Value*

For more details see *Predictive power criteria* [↗](#) (page 3190).

METHOD=C4.5

When `METHOD=C4.5`, there is no explicit `CRITERION` option available, because *Entropy Gain Ratio* is always used as the criterion.

METHOD=CART

When `METHOD=CART`, the following `CRITERION` options are available:

- `CRITERION=GINI` uses *Gini Variance*
- `CRITERION=ORDEREDTWOING` uses *Ordered Twoing*
- `CRITERION=TWOING` uses *Twoing*
- `CRITERION=LEASTSQUAREDDEVIATION` uses *Least Squared Deviation*

Predictive power criteria

Predictive power is a way of measuring how well a particular input variable can predict the target variable.

Pearson's Chi-Squared statistic

Pearson's Chi-squared statistic is a measure of the likelihood that the value of the target variable is related to the value of the predictor variable.

Each observation in the dataset is allocated to a cell in a contingency table, according to the values of the predictor and target variables. Pearson's Chi-squared statistic is calculated as the normalised sum of the squared deviations between the actual number of observations in each cell, and the expected number of observations in each cell if there were no relationship between the predictor and target variables.

If a predictor variable has a high Pearson's Chi-squared statistic, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Pearson's Chi-squared statistic for a discrete target variable is calculated as

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}}$$

where:

- N is the total number of observations in the dataset
- r is the number of distinct values of the predictor variable X (these are the rows in the contingency table)
- c is the number of distinct, discrete values of the target variable Y (these are the columns in the contingency table)
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j (these are the values in the cells of the contingency table)
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- μ_{ij} is the expected value of n_{ij} , calculated as

$$\mu_{ij} = \frac{n_{i*}n_{*j}}{N}$$

Entropy Variance

Entropy variance is a measure of how well the value of a predictor variable can predict the value of the target variable.

If a variable in a dataset has a high entropy variance, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Entropy variance for a discrete target variable is calculated as

$$E_r = 1 - \frac{\sum_{i=1}^r \left(\frac{n_{i*} E_i}{N} \right)}{E}$$

where:

- N is the total number of observations in the dataset
- r is the number of distinct values of the predictor variable, X
- c is the number of distinct, discrete values of the target variable, Y
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- E_i is the entropy calculated for just the observations where the predictor variable is X_i , calculated as

$$E_i = - \frac{1}{\log(c)} \sum_{j=1}^c \frac{n_{ij}}{n_{i*}} \log \left(\frac{n_{ij}}{n_{i*}} \right)$$

- E is the entropy calculated for all the observations, calculated as

$$E = - \frac{1}{\log(c)} \sum_{j=1}^c \frac{n_{*j}}{N} \log \left(\frac{n_{*j}}{N} \right)$$

Gini Variance

Gini variance is a measure of how well the value of a predictor variable can predict the target variable.

If a variable in a dataset has a high Gini variance, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Gini variance for a discrete target variable is calculated as

$$G_r = 1 - \frac{\sum_{i=1}^r \left(\frac{n_{i*} G_i}{N} \right)}{G}$$

where

- N is the total number of observations in the dataset
- r is the number of distinct values of the predictor variable, X

- c is the number of distinct, discrete values of the target variable, Y
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- G_i is the Gini impurity calculated for just the observations where the predictor variable is X_i , calculated as

$$G_i = 1 - \frac{\sum_{j=1}^c n_{ij}^2}{n_{i*}^2}$$

- G is the Gini impurity calculated for all the observations, calculated as

$$G = 1 - \frac{\sum_{j=1}^c n_{*j}^2}{N^2}$$

Information value

Information value is a measure of the likelihood that the value of the target variable is related to the value of the predictor variable. The information value measure is only applicable for binary target variables (that is, target variables that can take one of exactly two values).

If a predictor variable has a high information value, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

The information value statistic is calculated as

$$IV = \sum_{i=1}^r \left(\frac{n_{i0}}{n_{*0}} - \frac{n_{i1}}{n_{*1}} \right) WOE_i$$

where:

- r is the number of distinct, discrete values of the predictor variable X (these are the rows in the contingency table)
- Y_0 and Y_1 are the two possible values of the binary target variable Y
- n_{i0} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the value, Y_0 (these are the values in the cells of the Y_0 column in the contingency table)
- n_{i1} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the value, Y_1 (these are the values in the cells of the Y_1 column in the contingency table)
- n_{*0} is the total number of observations for which the target variable Y has the value, Y_0
- n_{*1} is the total number of observations for which the target variable Y has the value, Y_1

- $\alpha \ll 1$ is the weight of evidence (WOE) adjustment, a small positive number to avoid infinite values when $n_{i0} = 0$ or $n_{i1} = 0$
- WOE_i is the WOE value for observations where the predictor variable is X_i , calculated as

$$WOE_i = \ln\left(\frac{n_{i0}}{n_{*0}} + \alpha\right) - \ln\left(\frac{n_{i1}}{n_{*1}} + \alpha\right)$$

Using the DECISIONTREE procedure

This example shows how to use PROC DECISIONTREE to build a decision tree from an input dataset and to measure how well this tree can predict the target variable.

This example uses a simple dataset containing a sample of 100 people and their age, salary, make of car, whether they own a dog and whether they have defaulted on a loan. In this example, the DECISIONTREE procedure is used to generate a decision tree using the C4.5 algorithm.

Dataset

This example uses the dataset `loanData`, which contains the following observations:

Age	Salary	Car	Dog	Loan_Default
21	21325	Ford	1	0
21	30154	Ford	0	0
21	52389	Ford	0	1
22	59703	Ford	0	0
22	34264	Ford	0	1
22	9720	.	0	0
22	43123	Ford	0	0
22	65111	Ford	0	0
23	48437	Ford	0	0
24	3748	.	0	0
24	42226	Ford	0	1
24	36632	Ford	0	1
25	48310	Ford	1	0
25	27238	Ford	1	0
26	25927	Ford	0	0
26	59457	Nissan	0	0
26	39058	Nissan	0	0
27	66886	Nissan	0	1
28	62063	Nissan	1	0
29	55120	Nissan	0	0
30	67674	Nissan	0	0
32	7598	Ford	0	0
33	15708	Ford	1	0
33	53192	Nissan	0	0
33	44778	Nissan	1	0
34	9123	Ford	0	0
36	93027	Volvo	0	0
36	53889	Volvo	0	0
37	106263	Volvo	0	0
41	44477	Volvo	0	1
41	34316	Nissan	1	1

42	68092	Volvo	1	0
42	59812	Volvo	0	0
42	109801	Volvo	0	1
43	67401	Volvo	0	0
43	119848	Volvo	0	0
43	29937	Ford	0	0
43	83910	Volvo	0	0
44	69805	Volvo	0	0
44	10185	Ford	0	0
44	70349	Volvo	0	1
45	108497	Volvo	1	0
45	18964	Ford	1	0
45	63852	Volvo	0	0
45	60078	Volvo	1	0
46	110470	VW	0	0
46	94727	VW	0	0
46	120335	VW	0	0
47	135657	VW	0	1
47	117581	VW	0	0
47	124175	VW	0	0
47	21844	Ford	0	1
47	23676	Ford	0	1
48	80039	VW	0	0
48	49712	Volvo	0	0
49	42996	Volvo	1	0
50	24927	Ford	0	0
50	143032	VW	1	0
50	3377	Ford	0	0
51	100965	BMW	0	1
52	109383	BMW	0	0
52	152770	BMW	1	0
52	101555	BMW	0	0
52	95710	VW	0	0
52	147672	BMW	1	1
54	52246	Volvo	0	0
54	81418	VW	0	0
54	22060	Ford	0	1
54	115577	BMW	0	0
55	59722	Volvo	0	1
56	164395	BMW	0	0
56	157190	BMW	0	1
56	17268	Ford	0	0
56	60255	Volvo	0	0
56	162720	BMW	0	0
58	163492	BMW	0	0
59	114415	BMW	0	1
59	134879	BMW	0	0
59	33462	Nissan	0	1
60	7823	Ford	0	0
60	134460	BMW	0	0
61	773	Ford	0	0
64	30254	Nissan	0	0
64	154860	BMW	0	0
64	58275	Volvo	0	0
64	122884	BMW	0	0
65	138848	BMW	0	0
65	123111	BMW	1	0
66	147121	BMW	0	0
66	188638	BMW	0	0
66	165768	BMW	1	0

66	89132	VW	0	1
66	144425	BMW	0	0
68	91219	VW	0	0
68	193010	BMW	0	0
68	39799	Nissan	1	0
68	41234	Volvo	0	0
69	33319	Nissan	1	1
69	180753	BMW	1	0
70	16219	Ford	1	0

Code example

The model defines `age`, `salary`, and `car` as input variables. `age` is an ordinal variable (it contains discrete values with an implicit ordering), `salary` is an interval variable (it can be regarded as continuous) and `car` is a nominal variable (it contains discrete values with no implicit ordering). The target variable (the variable that the decision tree needs to predict) is `loan_default`. The decision tree algorithm used is C4.5. The criterion used to split nodes in the tree is `ENTROPYVAR` (the predictive power using this split is measured using *Entropy Variance*) which is the default criterion for `METHOD = C4.5`, and in fact, the only valid criterion for this method. The output shows the successful and unsuccessful classifications using the decision tree generated from these parameters.

```
PROC DECISIONTREE
  DATA = loanData
  MAXDEPTH=5
  OUTTREE = LoanDataTree
  METHOD = C4.5;
INPUT salary /LEVEL=INTERVAL;
INPUT age /LEVEL=ORDINAL;
INPUT car /LEVEL=NOMINAL;
TARGET loan_default/LEVEL=BINARY;
RUN;
```

Target Summary

The Target Summary table shows the target type that the decision tree is required to predict. In this case, it's a discrete target with binary (yes or no) values.

Discrete or continuous?	Level type	Target Summary		
		Number of categories	Order type	Descending?
DISCRETE	BINARY	2	INTERNAL	NO

Input Summary

The Input Summary table summarises the input variables, their types, categories and the other values that control the decision tree to build.

Run Summary

The Run Summary table summarises information about the run, for example, input dataset, the decision tree parameters that were used, the number of data items processed, and information about the decision tree that was generated (the number of nodes and the depth of the tree).

Classification

The Classification table shows the *confusion matrix*, where the rows represent the actual categories, and the columns represent categories predicted by the decision tree. The table entries show how many observations of each category were mapped to each of the predicted categories, so the leading diagonal contains the numbers of observations that were predicted correctly, and the other entries in the table contain the observations that were misclassified.

In this example, 77 non-defaulters (actual category 0) were correctly classified, but three non-defaulters were classified as category 1 (likely to default). 11 defaulters (actual category 1) were correctly classified as likely to default, but 10 were incorrectly classified as unlikely to default.

Predicted Category Frequencies			
Actual Category	0	1	
0	77	2	
1	10	11	

Scoring another dataset

This decision tree could then be used on another dataset to predict classifications from the predictor variables. To do this, use the `CODE` statement to save the decision tree as a code file, then specify the code file as an `%INCLUDE` in a data step.

DECISIONTREE procedure reference

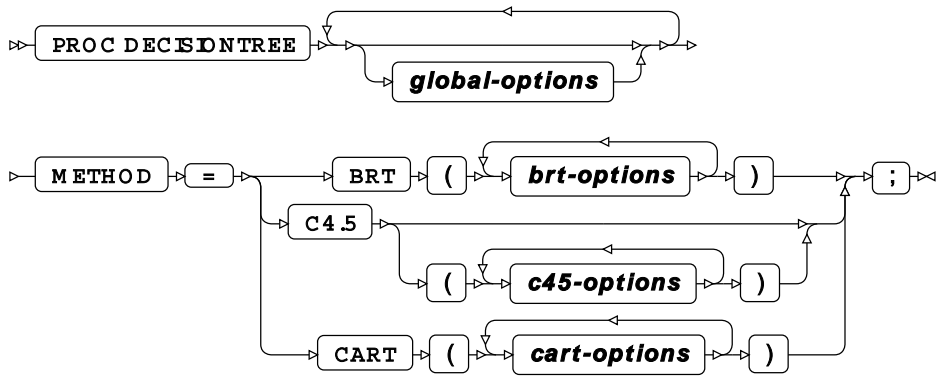
Describes the syntax and options for `PROC DECISIONTREE` and its contained statements.

<code>PROC DECISIONTREE</code> ↗	3197
Specifies the algorithm and options to use to create a decision tree from the input dataset.	
<code>CODE</code> ↗	3204
Outputs the decision tree as a file containing data step code, which can subsequently be used to score another dataset using this decision tree.	
<code>FREQ</code> ↗	3205
Specifies a variable containing the frequency associated with an observation.	
<code>INPUT</code> ↗	3205
Specifies the input (predictor) variables and the options to use for each.	
<code>TARGET</code> ↗	3207
Specifies the target (dependent) variable and any options that apply to the variable.	

WHERE [↗](#)..... 3209
Restricts the observations to be processed.

PROC DECISIONTREE

Specifies the algorithm and options to use to create a decision tree from the input dataset.



The `DECISIONTREE` procedure is a machine learning algorithm that creates a tree-like decision model from the specified input dataset using the specified options. The `METHOD` option is mandatory and defines the algorithm to use to construct the tree (one of BRT, C4.5 or CART). There are also some global options that can be specified for any kind of decision tree, and some algorithm-specific options.

global-options

Specifies the global options for any kind of decision tree. See [Global options ↗](#) (page 3197).

brt-options

Specifies the algorithm options for binary response trees. See [BRT options ↗](#) (page 3199).

c45-options

Specifies the algorithm options for C4.5 decision trees. See [C4.5 options ↗](#) (page 3201).

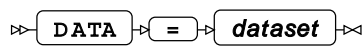
cart-options

Specifies the algorithm options for CART decision trees. See [CART options ↗](#) (page 3202).

Global options

The following *global-options* are available:

DATA



Specifies the training dataset used to construct the decision tree. If a training dataset is not specified, the most recently-created dataset is used.

EXCLUDEMISS

» **EXCLUDEMISS** «

Specifies that observations containing missing values are excluded when determining the best split at a node. By default, observations with missing values are included.

MAXDEPTH

» **MAXDEPTH** = *depth* «

Specifies the maximum depth of the tree. By default, the tree depth is unlimited.

MINNODESIZE

» **MINNODESIZE** = *size* «

Specifies the minimum number of observations per decision tree node. If a proposed split would create a node containing fewer than the specified minimum number of observations, the split does not occur.

The default value is determined by **METHOD**. When **METHOD** = **BRT** is specified, the default is 200. When **METHOD=C4.5** is specified, the default is 2. When **METHOD=CART** is specified, the default is 1.

MINNODESIZERATIO

» **MINNODESIZERATIO** = *ratio* «

Specifies the minimum size of a node as a percentage of the observations in the dataset. If a proposed split would create a node containing fewer than the specified percentage of observations, the split does not occur. The default value is 5 (percent).

NOPRINT

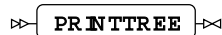
» **NOPRINT** «

Specifies that all ODS output is suppressed.

OUTTREE

» **OUTTREE** = *dataset* «

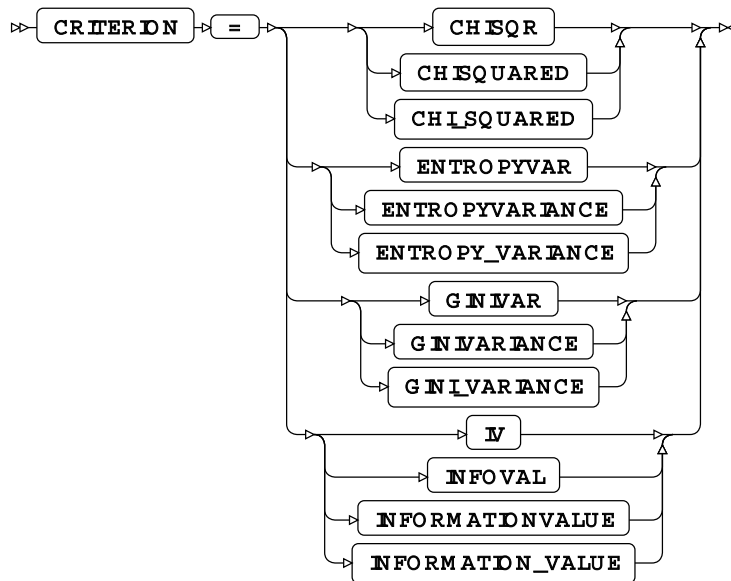
Specifies that the decision tree structure is written to the specified output dataset.

PRINTTREE

Specifies that the decision tree structure is written to ODS output.

BRT options

The following *brt*-options are available when `METHOD=BRT`:

CRITERION

Specifies the criterion used to split nodes in the decision tree.

When `METHOD=BRT`, this option is mandatory and must be specified. The following values are available:

CHISQR

Specifies that *Pearson's Chi-Squared* statistic is used to measure the predictive power of variables.

ENTROPYVAR

Specifies that *Entropy Variance* is used to measure the predictive power of variables.

GINIVAR

Specifies that *Gini Variance* is used to measure the predictive power of variables by measuring the strength of association between variables.

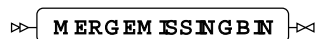
IV

Specifies that *Information Value* is used to measure the predictive power of variables.

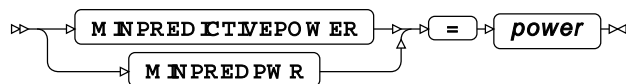
MAXPREDICTIVEPOWERCHANGE

Specifies the maximum change allowed in the predictive power when optimally merging bins.

The default value is determined by the type of `CRITERION`. When `CRITERION = CHISQR` is specified, the default is 0.002. When `CRITERION=INFOVAL` is specified, the default is 0.01. For all other settings of `CRITERION`, the default is 0.001.

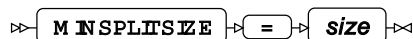
MERGEMISSINGBIN

Specifies that missing values are considered a separate valid category when binning data.

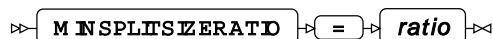
MINPREDICTIVEPOWER

Specifies the minimum predictive power required for a decision tree node to be split.

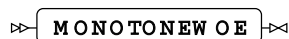
The default value is determined by the type of `CRITERION` specified: when `CRITERION = CHISQR`, the default is 0.004; when `CRITERION = INFOVAL`, the default is 0.02; for all other settings of `CRITERION`, the default is 0.002.

MINSPLITSIZE

Specifies the minimum number of observations a decision tree node must contain in order for it to be split further. If the node contains fewer observations than the specified minimum split size, the node is not split. The default `MINSPLITSIZE` is 1000.

MINSPLITSIZERATIO

Specifies the minimum number of observations a decision tree node must contain for the node to be split, as a percentage of the observations in the dataset. If the node contains fewer observations than the percentage specified as the minimum split size, the node is not split. The default value is 2 (percent).

MONOTONEWOE

Ensures that the weight of evidence (WoE) value for ordered input variables is either monotonically increasing or monotonically decreasing.

NOALLOWSAMEVARSPILT

» **NOALLOW SAMEVARSPILT** «

Specifies that a variable cannot be used more than once to split a decision tree node.

NOOPENLEFT

» **NOOPENLEFT** «

Specifies that, for a continuous variable, the node containing the very lowest values (the node on the far left) has a closed lower bound. Otherwise, the lower bound of the node containing the very lowest values is $-\infty$ (minus infinity).

NOOPENRIGHT

» **NOOPENRIGHT** «

Specifies that, for a continuous variable, the node containing the very highest values (the node on the far right) has a closed upper bound. Otherwise, the upper bound of the node containing the very highest values is ∞ (infinity).

WOEADJUST

» **WOEADJUST** = *adjustment* «

Specifies the adjustment applied in weight of evidence calculations to avoid invalid results for pure inputs.

The default value for **WOEADJUST** is 1E-5.

C4.5 options

Note that C4.5 decision trees always use the *Entropy Gain Ratio* to split the nodes in the decision tree, so there is no **CRITERION** option when **METHOD=C4.5**.

The following *c45-options* are available when **METHOD=C4.5**:

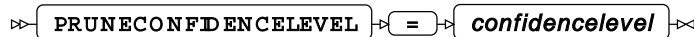
MERGE CATEGORIES

» **MERGE CATEGORIES** «

Specifies that discrete independent variables are grouped together to optimize the dependent variable value used for splitting.

PRUNE

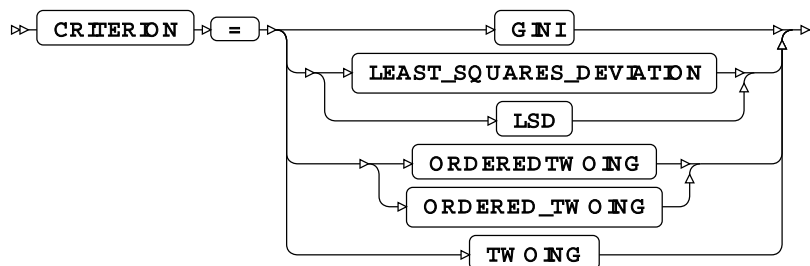
Specifies that the decision tree should be pruned to reduce tree complexity by removing nodes from a tree that do not significantly improve the predictive accuracy of the model. Pruning a tree may reduce the likelihood of overfitting the model to test data.

PRUNECONFIDENCELEVEL

Specifies the confidence level for pruning, expressed as a percentage. The default value is 25 (percent).

CART options

The following *cart-options* are available when `METHOD=CART`:

CRITERION

Specifies the criterion used to split nodes in the decision tree.

When `METHOD=CART`, this option is mandatory and must be specified. The following values are available:

GINI

Specifies that *Gini Impurity* is used to measure the predictive power of variables, by measuring the purity of the nodes.

LEAST_SQUARES_DEVIATION

Specifies that the *Least Squared Deviation* is used to measure the predictive power of variables.

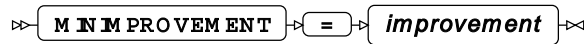
This option can only be specified for regression trees (that is, when `LEVEL=INTERVAL` is specified in the `TARGET` statement).

ORDEREDTWOING

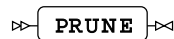
Specifies that the *Ordered Twoing Index* is used to measure the predictive power of variables.

TWOING

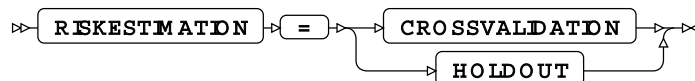
Specifies that the *Twoing Index* is used to measure the predictive power of variables.

MINIMPROVEMENT

Specifies the minimum improvement in impurity required for a split to occur.

PRUNE

Specifies that the decision tree should be pruned to reduce tree complexity by removing nodes from a tree that do not significantly improve the predictive accuracy of the model. Pruning a tree may reduce the likelihood of overfitting the model to test data.

RISKESTIMATION

Specifies the risk optimization method used during pruning. This option is ignored if `PRUNE` is not specified.

One of:

CROSSVALIDATION

The input dataset is divided as evenly as possible into ten randomly-selected groups. The analysis is repeated ten times, each time with a different group as test dataset, and the remaining groups used as training data. Risk estimates are calculated for each group and then averaged across all groups. The averaged risk values are used to prune the final tree built from the entire dataset.

This is the default value.

HOLDOUT

The input dataset is randomly divided into a test dataset containing one third of the dataset and a training dataset containing the balance of the dataset. The final tree is initially built using the training dataset, then pruned using risk estimation calculations on the test dataset.

CODE

Outputs the decision tree as a file containing data step code, which can subsequently be used to score another dataset using this decision tree.



When the code file has been produced, you can write a data step that contains a `SET` statement to specify the dataset to be scored, followed by a `%INCLUDE` statement to include the decision tree code file. The output dataset contains the scored data.

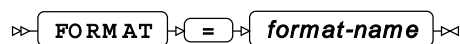
fileref

Specifies the name of the file to contain the code.

Options

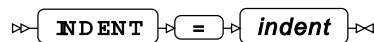
The following *options* are available:

FORMAT



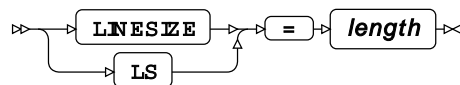
Specifies the number format to use for numeric values in the code file. Use this statement to ensure that decision tree parameters that are numeric are output at a sufficiently high resolution.

INDENT



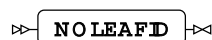
Specifies the number of spaces to use to indent blocks in the code file.

LINESIZE



Specifies the maximum line length before line wrapping occurs in the code file.

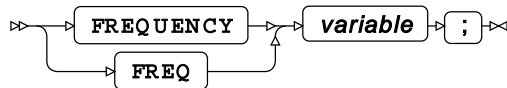
NOLEAFID



Specifies that the node IDs are not added to the nodes in the decision tree. If this option is not specified, a numeric node ID is added to each node.

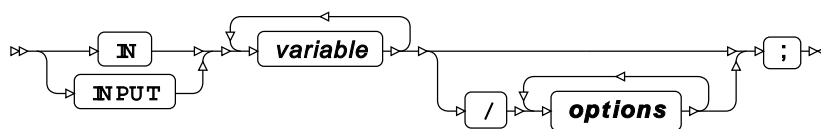
FREQ

Specifies a variable containing the frequency associated with an observation.



INPUT

Specifies the input (predictor) variables and the options to use for each.



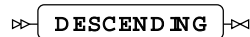
variable

A variable to which measures of predictive power are applied.

Options

The following *options* are available:

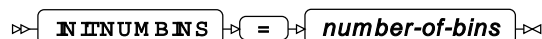
DESCENDING



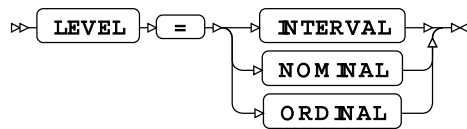
Specifies a descending sort order for the variable. This option only applies if `ORDER=FORMATTED` or `ORDER=INTERNAL` is also specified, as the sort order cannot be explicitly determined from these options. If not specified, the variable is assumed to be in ascending sort order.

If `ORDER` is not equal to `FORMATTED` or `INTERNAL` then the sort order for the variable is determined by the `ORDER` option.

INITNUMBINS



For `METHOD=BRT`, specifies the initial number of bins to use. The default initial number of bins is 50. The initial number of bins is in the range 2–100. This option only has an effect if `METHOD=BRT` is specified in the `PROC DECISIONTREE` statement.

LEVEL

Specifies the level for the input variables. The default `LEVEL` value is `INTERVAL`.

INTERVAL

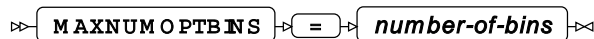
Specifies a continuous input variable with an implicit category ordering.

NOMINAL

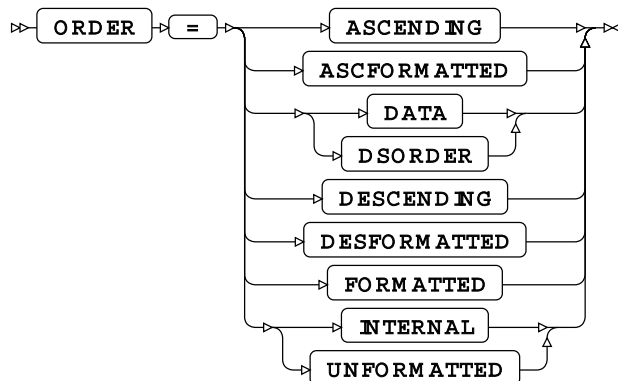
Specifies a discrete input variable with no implicit ordering. When partitioning this variable into nodes in the decision tree, any category can be merged with any other category.

ORDINAL

Specifies a discrete input variable with an implicit category ordering. When partitioning this variable into nodes in the decision tree, only adjacent categories can be merged together.

MAXNUMOPTBINS

For `METHOD=BRT`, specifies the maximum number of optimal bins to use. The default maximum number of bins is 10. The specified number must be between 2 and 100. This option only applies if `METHOD=BRT` is specified in the `PROC DECISIONTREE` statement.

ORDER

Specifies the order of the input variable. The default `ORDER` value is `INTERNAL`.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

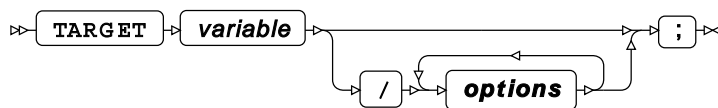
The variable is sorted by the formatted value. If the (separate) **DESCENDING** option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) **DESCENDING** option is also specified, the sort order is descending, otherwise, the sort order is ascending.

TARGET

Specifies the target (dependent) variable and any options that apply to the variable.



Only one **TARGET** statement is allowed in each **PROC DECISIONTREE** statement. The **TARGET** statement must contain a single variable, and the specified variable must correspond to a classification level.

variable

The dependent variable.

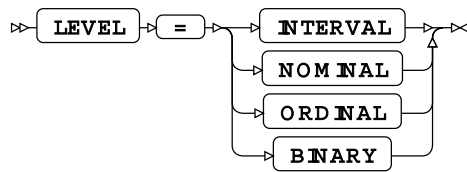
Options

The following *options* are available:

DESCENDING

Specifies a descending sort order for the variable. This option only applies if **ORDER=FORMATTED** or **ORDER=INTERNAL** is also specified, as the sort order cannot be explicitly determined from these options. If not specified, the variable is assumed to be in ascending sort order.

If **ORDER** is not equal to **FORMATTED** or **INTERNAL** then the sort order for the variable is determined by the **ORDER** option.

LEVEL

Specifies the level for the target variable. The default **LEVEL** value is **NOMINAL**.

INTERVAL

Specifies a continuous target variable containing an implicit category ordering.

NOMINAL

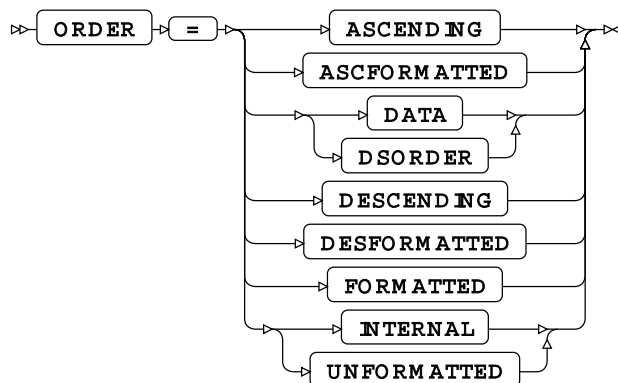
Specifies a discrete target variable with no implicit ordering.

ORDINAL

Specifies a discrete target variable with an implicit category ordering.

BINARY

Specifies a target variable that can take one of two values.

ORDER

Specifies the order of target variable. The default **ORDER** value is **INTERNAL**.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

The variable is sorted by the formatted value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

WHERE

Restricts the observations to be processed.

» **WHERE** » *condition* » ; »

GMM procedure

The GMM procedure enables you to build a Gaussian mixture model (GMM) from an input dataset. You can then use the GMM to analyse other datasets.

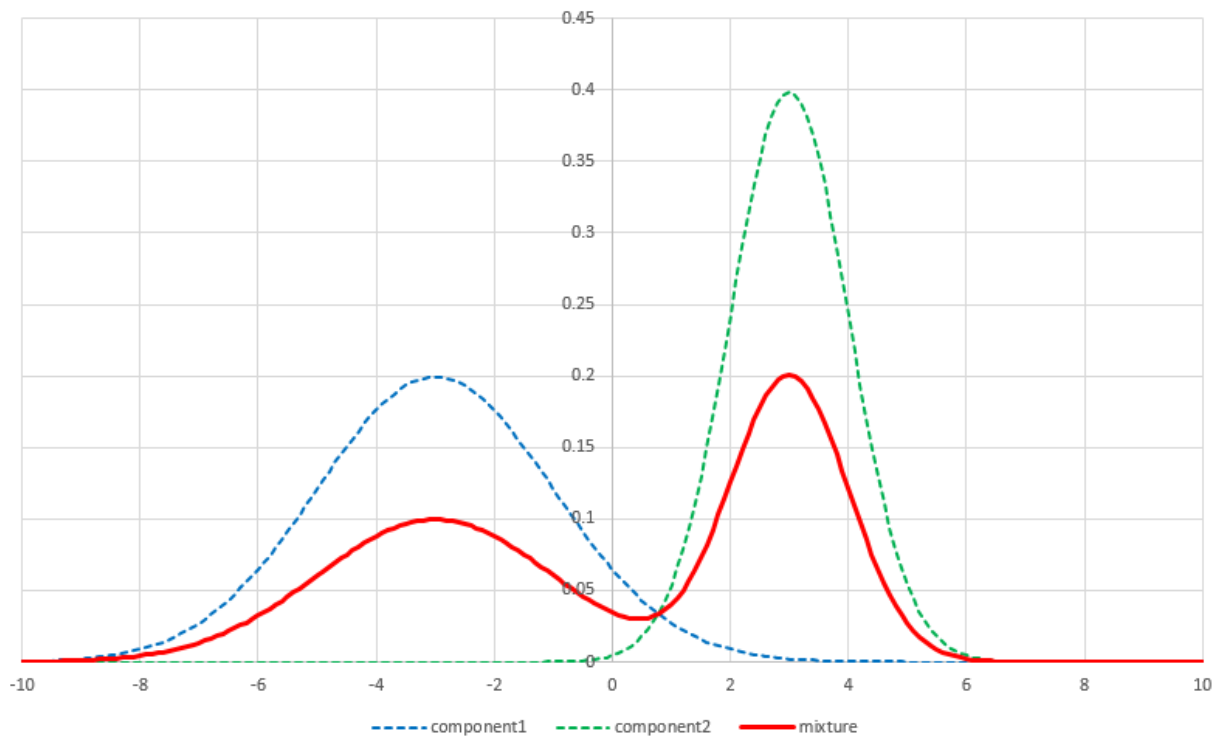
About Gaussian mixture models

A Gaussian mixture model (GMM) is a weighted combination of one or more Gaussian probability densities.

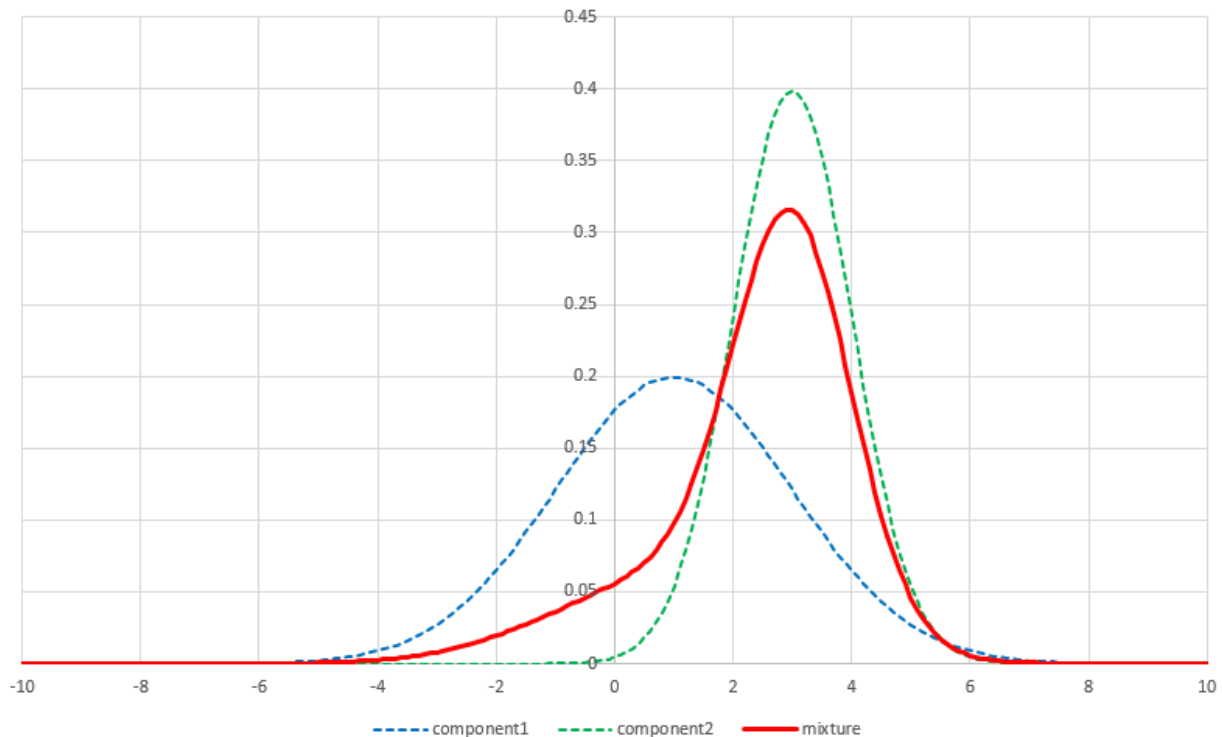
Introduction to Gaussian mixture models

Gaussian mixture models are capable of accurately modelling complex univariate or multivariate data distributions that cannot be modelled by a single distribution.

For example, a bimodal univariate distribution (shown in red in the plot below) can be modelled as a weighted sum of two Gaussian distributions with different means and standard deviations (shown in blue and green in the plot below). In this plot, each of the distributions contributes an equal weight to the mixture model.

Figure 371. Bimodal distribution represented as a combination of two Gaussian distributions

Similarly, a skewed unimodal univariate distribution (shown in red in the plot below) can be modelled as a weighted sum of two Gaussian distributions with different means and standard deviations. In this plot, one distribution (shown in blue) has a lower mean and a larger standard deviation, and has a weight of 0.3, so contributes to the long tail without affecting the modal value. The second distribution (shown in green) has a larger mean and smaller standard deviation, and has a weight of 0.7.

Figure 372. Skewed distribution represented as a combination of two Gaussian distributions

Complex multivariate distributions can be modelled in the same way, as a mixture of two or more Gaussian distributions.

Gaussian mixture models have many applications. As well as modelling complex probability density functions, Gaussian mixture models can identify clusters or categories of data, and determine which cluster or category an individual observation is most likely to belong to.

Gaussian mixture model parameters

For a multivariate distribution with d independent variables, a Gaussian mixture model is parametrised by:

- The number of components, k ; that is, the number of independent Gaussian distributions that contribute to the probability density function
- For each of the k components:
 - a d -dimensional vector containing the mean values of each of the d variables
 - a d by d precision matrix, which is the inverse of the covariance matrix for that component, and provides information about the partial correlation of each pair of variables.
 - a weight indicating the proportion which that component contributes to the overall density value.

In general, a Gaussian mixture model is generated from a dataset by:

- choosing initial values for the number of components, their means, covariances and weights
- iteratively refining these values to improve them

- terminating the process when a specified tolerance is reached or a specified number of iterations have been completed.

For more information about Gaussian mixture models and their parameters, see *Mixture models* [↗](#) (page 3215), *Gaussian mixture models* [↗](#) (page 3216) and *Posterior probabilities* [↗](#) (page 3216).

PROC GMM provides a number of options to control the way that the Gaussian mixture model parameters are derived.

Component initialisation

The INIT option allows you to choose various ways of deriving the initial values for the model parameters, depending on the prior information you have about the data distribution.

- If you know the number of components, you can use the PROC GMM K option to specify it. If you also know initial estimates for the means, covariances and weights of each component, you can specify those too, using INIT=CUSTOM or INIT=TABLE. Alternatively, PROC GMM can calculate initial values for each of the model parameters, either using the *k-means++* algorithm (INIT=KPP), or randomly (INIT=RAND). For more information about the k-means algorithm, see the section on k-means ++ in *Maximum likelihood and expectation maximisation* [↗](#) (page 3217).
- If you do not know the number of components in advance, you can also use the k-means++ algorithm or a random algorithm to calculate initial values for the model parameters. In this case, you must choose how the number of components is estimated. This process is called *model selection*, and the PROC GMM SELECT option controls this.

Model selection

The SELECT option allows you to specify how the initial values for the model parameters are selected if you do not know the number of components in the model. PROC GMM supports three different model selection methods:

- *The Akaike Information Criterion* (SELECT=AIC)
- *Bayesian Information Criterion* (SELECT=BIC)
- *Variational Bayes* (SELECT=VB).

SELECT=AIC and SELECT=BIC both require that you know the maximum possible number of components, KMAX, as both these options create multiple initial models, one for each of the possible numbers of components, then choose the best one. For each of the KMAX models, the initial parameter values are derived using the specified INIT method (KPP or RAND).

You can use SELECT=VB whether you know KMAX or not, although the options that need to be specified are different in each case.

- If you know KMAX, then you simply specify KMAX, and the prior hyperparameters, A0, B0, M0, NU0 and V0 (all of which have default values if preferred). The Variational Bayes approach creates a mixture model with KMAX components and uses the prior hyperparameters and prior probability distributions to determine the initial component weights. Components that appear insignificant have low weights. Initial values for the model parameters for each of the components are derived using the specified INIT method (KPP or RAND).

- If you do not know `KMAX`, then there is an additional prior step, to derive a suitable initial value for `KMAX`. In this case, you also need to specify values for `PRIORITERS`, `PRIORRUNS`, and `PRIORSAMPLES` (these also have default values you can use). The Variational Bayes approach uses these values to derive the initial number of components and the values for the prior hyperparameters. Then, as before, the initial values for the model parameters for each of the components are derived using the specified `INIT` method (`KPP` or `RAND`).

For more information about the model selection process, see *Model selection* [↗](#) (page 3218).

Component iteration

Once the initial values have been derived for the model parameters, they are iteratively refined to improve them. The way that `PROC GMM` does this depends whether you specified the number of components explicitly in the `INIT` option, or used the `SELECT` option because you didn't know `K`. If you used the `SELECT` option, the algorithm also depends on whether you chose the `AIC`, `BIC` or `VB` criteria.

- If you explicitly specified the number of components, `K`, the initial model parameters are successively refined using *expectation maximisation* (EM) until the specified termination condition is reached. For more information about expectation maximisation, see *Maximum likelihood and expectation maximisation* [↗](#) (page 3217).
- If you chose `SELECT=AIC` or `SELECT=BIC`, then each of the models from 1 to `KMAX` components are successively refined using EM until the specified termination condition is reached for each model. Then the specified criterion (`AIC` or `BIC`) is used to choose the best model from the set of models.
- If you chose `SELECT=VB`, the situation is slightly different. The Variational Bayes approach defines not only how to derive the initial values for the number of components and the model parameters, but also how to iteratively refine the estimates, until the specified termination condition is reached.

Iteration termination

The algorithm terminates when the convergence tolerance specified by `CTOL` is reached.

If EM is used to measure the improvement in the model at each iteration (`K` is known, or `SELECT=AIC`, or `SELECT=BIC`), the algorithm terminates when the absolute value of the change in the log likelihood function is less than the value specified in `CTOL`.

If the Variational Bayes approach is used to refine the model (`SELECT=VB`), the algorithm terminates when the absolute value of the change in the variational lower bound function is less than the value specified in `CTOL`.

Alternatively, if the specified maximum number of iterations, `MAXITERS`, have been completed and the model has not yet converged, the algorithm terminates.

Note that these algorithms are sensitive to the initial values of the model parameters. Even if the algorithm appears to converge, the final model parameters may only represent a local maximum, rather than the overall optimal maximum.

Covariance matrices

The covariance matrix is a d by d matrix that contains the variances of each of the variables on the leading diagonal, and the covariances between each pair of variables in the other positions. It is symmetric about the leading diagonal. `PROC GMM` supports the following options for covariance matrices:

- `COVARIANCE=FULL`: the covariance matrices are symmetric about the leading diagonal and any value may be zero or non-zero.
- `COVARIANCE=DIAGONAL`: the off-diagonal elements of the covariance matrices are all equal to zero.
- `COVARIANCE=SPHERICAL` : the covariance matrices are diagonal, and additionally, every element is the same.

`PROC GMM` requires that all covariance matrices are positive definite. In order to ensure this, you can optionally use the `REG` sub-option to specify a regularisation value to add to the diagonal elements of the covariance matrices to try to ensure that they are positive definite.

You can optionally use the `TIED` sub-option to specify that all components have the same covariance matrix.

Outputs

By default, `PROC GMM` produces a Fit Summary table and a Model Parameters table using ODS output. The Fit Summary table contains the options specified to generate the model, the number of components in the optimal model and some values that determine how well the derived model fits the data: the log likelihood, AIC, BIC and (if appropriate) variational lower bound values. The Model Parameters table contains the model parameters for each component in the mixture model.

If `SELECT=AIC` or `SELECT=BIC`, you can also specify the `SUMMARY` option to output the details of all the intermediate models that were considered.

Once you have generated a Gaussian mixture model, you can use the `PROC GMM OUTEST` option to save the model parameters in a dataset. You can then use `PROC GMM INEST` to initialise the model generation process for another model with similar distribution parameters, or to continue refining the original model.

You can also use the `PROC GMM CODE` statement to save the code to define the mixture model as a file that can be included in a data step. This can be used to score another dataset against the saved mixture model.

Scoring models

Once you have created a Gaussian mixture model for a dataset, you can use it to score that dataset or another dataset. The `SCORE` statement takes the data in the specified dataset and scores it using the model defined in `PROC GMM`. The score results are saved in a table which can be printed or saved in an output dataset. For each observation in the dataset being scored, the score results table shows the component that the observation is most likely to belong to, and the overall probability density of the distribution at that point.

You can use the `SCORE PRINT` option to write the score table to ODS output, and `SCORE OUT` to save the score results in a dataset.

For datasets with one or two variables, you can also output ODS plots using `SCORE PLOTK` to output a density plot for each component and `SCORE PLOTP` to output a mixture density plot for the score dataset.

Mixture models

We are given a dataset $D = \{\vec{x}_1, \dots, \vec{x}_n\}$ where each \vec{x}_i is a d -dimensional vector of real numbers. It is assumed that the points are generated in an independent and identically distributed (IID) manner from an underlying density $p(\vec{x})$. We further assume that $p(\vec{x})$ is defined as a finite mixture model with components:

$$p(\vec{x}|\Theta) = \sum_{j=1}^K \pi_j p(\vec{x}|z_j, \theta_j)$$

Where:

- z_j is the j^{th} element of a hidden, or latent, variable (\vec{z}) that is a K -dimensional vector of ones and zeroes (where the sum of the elements is one). If the value of the j^{th} element is one, then this indicates that the j^{th} component was selected to generate the observed sample at \vec{x} .
- $p(\vec{x}|z_j, \theta_j)$ is the j^{th} mixture component and is the density, or distribution, defined over $p(\vec{x})$, with parameters θ_j .
- π_j is a mixture parameter that represents the probability that a randomly-selected \vec{x} was generated by the j^{th} component. Note that:

$$\sum_{j=1}^K \pi_j = 1$$

$$\pi_j = (z_j = 1)$$

- Θ is the complete set of parameters for the mixture model:

$$\Theta = \{\pi_1, \dots, \pi_n, \theta_1, \dots, \theta_n\}$$

Gaussian mixture models

With a Gaussian mixture model, each of the K components is considered to be a multi-variate normal density with parameters $\vec{\mu}_j$ (a d -dimensional mean vector) and Λ_j (a d by d precision matrix, which is the inverse of the covariance matrix). The multi-variate normal density is defined as:

$$p(\vec{x}|\theta_j) = N(\vec{x}|\vec{\mu}_j, \Lambda_j) = \left(\frac{|\Lambda_j|}{(2\pi)^d} e^{-\left(\vec{x}-\vec{\mu}_j\right)^T \Lambda_j (\vec{x}-\vec{\mu}_j)} \right)^{\frac{1}{2}}$$

where

- $\theta_j = \{\vec{\mu}_j, \Lambda_j\}$ is the parameter set for the j^{th} component.
- $|\Lambda_j|$ is the determinant of the j^{th} precision matrix.

Our Gaussian mixture model density is:

$$p(\vec{x}|\Theta) = \sum_{j=1}^K \pi_j N(\vec{x}|\vec{\mu}_j, \Lambda_j)$$

Posterior probabilities

With mixture models we can compute the posterior probability $p(j|\vec{x})$ (a sample at \vec{x} has been generated by sampling from the distribution of the j^{th} mixture component) in a straightforward manner by using Bayes' theorem (Bishop, 2006 [3]):

$$P(j|\vec{x}) = \frac{p(\vec{x}|z_j=1)p(z_j=1)}{p(\vec{x})} = \frac{\pi_j N(\vec{x}|\vec{\mu}_j, \Lambda_j)}{\sum_{l=1}^K \pi_l N(\vec{x}|\vec{\mu}_l, \Lambda_l)}$$

This posterior probability can be thought of as the membership weight of the sample at \vec{x} in component j (given the set of model parameters, Θ). Such weights reflect our uncertainty, given the sample at \vec{x} and Θ , as to which of the K components generated the sample.

We assume that only one of the K components was used to generate the sample. Clipping small posterior probability values to zero can be a useful technique for speeding up the convergence of algorithms such as expectation maximisation.

For example, in the `PROC GMM` statement, you could specify `PPTOL=0.01`. This states that posterior probability values below 0.01 are clipped to zero.

Maximum likelihood and expectation maximisation

The *expectation maximisation* (EM) algorithm (Dempster, et al., 1977 [4]) starts from an initial estimate of the parameter set, Θ . It then proceeds to iteratively update Θ until convergence is detected.

To estimate the parameters of the model we use maximisation of the log-likelihood:

$$\Theta_{ML} = \underset{\Theta}{\operatorname{argmax}} [\ln(\mathbf{D}|\Theta)]$$

Assuming the samples in our dataset, \mathbf{D} , are independent, identically distributed (IID) random variables, the likelihood is defined as:

$$p(\mathbf{D}|\Theta) = \prod_{i=1}^n p(\vec{x}_i|\Theta) = \prod_{i=1}^n \sum_{j=1}^K \pi_j N(\vec{x}_i|\vec{\mu}_j, \Lambda_j)$$

So the log-likelihood is:

$$\ln p(\mathbf{D}|\Theta) = \sum_{i=1}^n \ln \left(\sum_{j=1}^K \pi_j N(\vec{x}_i|\vec{\mu}_j, \Lambda_j) \right)$$

Initialisation

If custom initialisation is selected, the procedure uses the supplied initial values for the component means, covariances and weights. Otherwise, the procedure uses the covariances of the data as an initial state for the covariances of each of the components and the inverse of the number of components for the initial mixing weights. Component means can either be selected randomly or using the k-means++ algorithm.

Random

If random initialisation is chosen, initial component means are chosen at random from the observations in the dataset.

K-means++

The *k-means++ algorithm* (Arthur & Vassilvitskii, 2007 [2]) is a heuristic method for choosing the initial component means. This algorithm has the following steps:

1. Choose the first mean vector randomly from one of the samples in the dataset using a uniform distribution:

$$p(\vec{\mu}_0 = \vec{x}_i) = U\{1, n\}$$

2. For subsequent mean vectors proceed in the following way:

- a. Calculate the square of the Mahalanobis distance (measured using an Euclidean distance) between a sample and the nearest component mean that has already been chosen ($D(\vec{x}_i)^2$). Compute these distances for each of the samples in the dataset.

- b. The i^{th} sample in the dataset is chosen as the next mean vector with probability:

$$p(\vec{\mu}_{next} = \vec{x}_i) = \frac{D(\vec{x}_i)^2}{\sum_{l=1}^n (\vec{x}_i)^2}$$

The expectation and maximisation steps

The expectation maximisation (EM) algorithm is iterative. It has an expectation step, and a maximisation step.

The expectation step calculates the posterior probabilities of component membership for each of the samples in the dataset, \mathbf{D} :

$$w_{ij}^{(t)} = P(j|\vec{x}_i) = \frac{\pi_j N(\vec{x}_i | \vec{\mu}_j^{(t)}, \Lambda_j^{(t)})}{\sum_{l=1}^K \pi_l N(\vec{x}_i | \vec{\mu}_l^{(t)}, \Lambda_l^{(t)})}$$

This formula is used to create an n by K matrix, \mathbf{W} , of the posterior probabilities, where n is the number of observations in the dataset and K is the number of components in the model. Then the values of the \mathbf{W} matrix are used to create a set of statistics for each component.

The maximisation step updates the parameters to maximise the expected value of the complete log-likelihood with respect to the posterior probability. The statistics from the expectation step are used to update the component parameters.

Convergence

The EM algorithm has converged when the log-likelihood function value is no longer increasing significantly; that is, the absolute change after an iteration of the algorithm is less than a specified tolerance.

For example, in the `PROC GMM` statement, you could specify `LLTOL=0.01`. This states that the EM algorithm has converged when the change in the calculated log-likelihood values is less than 0.01.

Model selection

The model selection step is required when K , the number of components in the model, is not known.

The model selection process can use the Akaike Information Criterion (AIC) (Akaike, 1973 [1]), the Bayesian Information Criterion (BIC) (Schwarz, 1978 [6]) or a Variational Bayes approach. AIC and BIC can only be used when the maximum possible number of components, K_{\max} , is known. The Variational Bayes approach can be used whether K_{\max} is known or not.

AIC and BIC metrics

In order to find the most appropriate value for K we try a range of K values from one to K_{\max} . We create a Gaussian mixture model for each value of K and use the selected metric (AIC or BIC) to determine the optimal value of K . The model with the value of K that gives the lowest value for the selected metric is the model that is chosen.

The Akaike Information Criterion formula is:

$$2(m_* - \ln(p(\mathbf{D}|\Theta)))$$

The Bayesian Information Criterion formula is:

$$m_* \ln(n) - 2 \ln(p(\mathbf{D}|\Theta))$$

In these formulae, m_* is the total number of free parameters in the model. This value depends on the number of components, the number of predictor variables and the type of precision matrix.

For example, in the `PROC GMM` statement, you could specify `SELECT=AIC (KMAX=10)`. This states that the AIC metric is used to evaluate models and that mixture models with all numbers of components between one and ten are evaluated.

Variational Bayes approach

In the Variational Bayes (approximate inference) approach, we give prior distributions to all unknown parameters, which are absorbed into our set of random variables. In some cases, it is difficult to evaluate the posterior distribution of these variables. So we use an approximation scheme, which either uses the expectations of the distribution, or uses a number of samples drawn from the distribution. We look to choose a variational distribution, q , from a family of distributions, such that we maximise the *variational lower bound* (Bishop, 2006 [3]).

Hyperparameters

In the Variational Bayes approach, the component parameters are conditioned on prior distributions (the Dirichlet distribution for the component weights, the multivariate normal distribution for the component means and the Wishart distribution for the component precision matrices). The parameters of these prior distributions are the component hyperparameters, as follows:

- α is the concentration weight and is used in the Dirichlet distribution to determine the component mixing weight (specified as `A0` in the `SELECT` option)
- β is the mean precision and is used in the multivariate normal distribution to determine the component mean (specified as `B0` in the `SELECT` option)
- \vec{m} is the expected mean and is used in the multivariate normal distribution to determine the component mean (specified as `M0` in the `SELECT` option)
- ν is the degrees of freedom and is used in the Wishart distribution to determine the component precision (specified as `NU0` in the `SELECT` option)

- \mathbf{V} is the scale parameter and is used in the Wishart distribution to determine the component precision (specified as `V0` in the `SELECT` option)

$$\{\pi_1, \dots, \pi_K\} \sim \text{Dir}(\alpha_1, \dots, \alpha_K)$$

$$\forall j \in \{1, \dots, K\} : \vec{\mu}_j \sim \mathcal{N}(\vec{m}_j, \beta_j \mathbf{\Lambda}_j)$$

$$\forall j \in \{1, \dots, K\} : \mathbf{\Lambda}_j \sim \mathcal{W}(\mathbf{V}_j, \nu_j)$$

The expected values for the component parameters are:

$$\forall j \in \{1, \dots, K\} : \mathbb{E}[\pi_j] = \frac{\alpha_j}{\sum_{l=1}^K \alpha_l}$$

$$\forall j \in \{1, \dots, K\} : \mathbb{E}[\vec{\mu}_j] = \vec{m}_j$$

$$\forall j \in \{1, \dots, K\} : \mathbb{E}[\mathbf{\Lambda}_j] = \nu_j \mathbf{V}_j$$

The hyperparameters for each component partly depend on prior hyperparameters that are either provided by the user or are derived from the data. These prior hyperparameters are denoted as $\{\alpha_0, \beta_0, \vec{m}_0, \nu_0, \mathbf{V}_0\}$.

For example, in the `PROC GMM` statement, you could select the Variational Bayes approach, and specify the following prior hyperparameters.

```
PROC GMM
  SELECT = VB (
    A0 = 1.0 B0 = 1.0 KMAX = 5 M0 = (0.0 0.0) NU0 = 2.0 V0 = (1.0 0.0 0.0 1.0)
  );
```

Responsibilities

We use the component hyperparameters to work out the responsibilities. A responsibility indicates how responsible a component is for generating a sample. The responsibilities, in turn, are used to update the hyperparameters (which are then used to update the component parameters).

Expectations or samples

To update the component parameters, we can either use the means of the prior distributions or draw a number of samples from the distribution and use the average of those. To use the means of the prior distributions (the default option), set `PARMSAMPLES=0` in the `SELECT` option: otherwise, if `PARMSAMPLES` is non-zero, that number of samples are drawn from the distribution instead.

For example, in the `PROC GMM` statement, you could specify `SELECT=VB (KMAX=32)`. This states that an expectation approach is used for finding the best model and that there are a maximum of 32 components.

To specify that sampling is used to find the best model, using five samples from the dataset for each parameter, and that there are a maximum of ten components, use `SELECT=VB (KMAX=10 PARMSAMPLES=5)`.

Convergence

The Variational Bayes algorithm has converged when the value of the variational lower bound is no longer changing significantly from one iteration to the next.

Infinite number of components

The mixture model may have an infinite number of components (Rasmussen, 2000 [5]) (Wood & Black, 2012 [7]) but only a finite number of these components have been used to generate the dataset, **D**. In this case, in the Variational Bayes approach, instead of specifying the maximum number of components, there is a prior processing stage to determine the number of components and the prior hyperparameters. The prior processing stage is a multi-part iterative process.

- In the first part of each iteration we update the prior hyperparameters using a sampling approach based on prior distributions, the statistics of the data and the current set of component parameters.
- In the second part, we generate new components from the new prior hyperparameters. New components are generated by sampling from the Dirichlet, multivariate normal and Wishart distributions using the latest set of prior hyperparameters. Then we probabilistically assign each sample to a component. There is a probability that the sample will be assigned to a newly created component (Rasmussen, 2000 [5]). After assigning samples we remove any components that no longer have any samples assigned to them. In this scheme we start off with just one component. The number of components we end up with will depend on the structure of the data in the dataset, the evolution of the parameters and the results of the assignments at each iteration.
- Finally, we use the set of components and prior hyperparameters from the first two parts to find a set of parameters and hyperparameters for the required components.

Once the prior processing stage is complete, and the maximum number of components and the prior hyperparameters are known, the procedure can maximise the value variational lower bound, as above.

For example, in the `PROC GMM` statement, you could specify `SELECT=VB (PRIORITERS=100 PRIORRUNS=1 PRIORSAMPLES=10)`. This states that the procedure uses the prior processing stage to determine the maximum number of components and the set of prior hyperparameters, using a single run with 100 iterations per run in the prior processing stage, and 10 samples whenever it updates a prior hyperparameter.

Using the GMM procedure

This example shows how to use `PROC GMM` to build a Gaussian mixture model from an input dataset and to measure how well this model represents the data.

Examples

The examples in this section illustrate two different ways that you can use `PROC GMM` to create a Gaussian mixture model from a training dataset.

Both examples use the publicly-available Iris dataset [8]. The Iris dataset consists of measurements of the widths and lengths of the petals and sepals of three species of Iris.

In these examples, the Iris dataset is used to derive a Gaussian mixture model for the probability distribution of two variables, iris petal lengths, and iris petal widths, for the three different species of iris, *setosa*, *versicolor* and *virginica*. Then the derived model is used to score the same dataset to see how well the predicted probability densities match the actual clusters of different kinds of iris. These examples only use two of the four measurements available in the dataset (sepal length and sepal width are also available) so that meaningful output plots can be produced.

Example (custom initialisation)

This example uses custom initialisation with a known number of components ($K=3$) to derive the parameters for the Gaussian mixture model. For each of these components, the `INIT` option supplies initial values for the mean petal length and petal width, initial covariance matrices giving the variances of each of the variables and the covariances between them, and the initial weights. The initial mean values were chosen by inspecting the data and choosing likely values, the initial covariance matrices are two by two identity matrices, and the initial mixture weights are equal.

The `VAR` statement specifies the predictor variables, `p_length` and `p_width`.

The `SCORE` statement uses the Gaussian mixture model to score the same dataset. The `PRINT` option includes the (already known) value of the `species` variable in the score table for comparison. Since there are only two variables in the multivariate distribution, density plots can be requested for each component (the `PLOTK` option) and for the overall mixture density (the `PLOTP` option).

```
PROC GMM
  DATA = MYLIB.iris
  COVARIANCE = FULL (REG = 0.000001)
  K = 3
  INIT = CUSTOM (
    MU = (1.5 0.3 4.5 1.3 5.5 1.8)
    SIGMA = (1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0)
    WT = (1.0 1.0 1.0))
  CTOL = 0.001;
VAR p_length p_width;
SCORE DATA = MYLIB.iris /
  OUT = OUTLIB.score_init_custom
  PRINT (var = species)
  PLOTK (OBS UNPACK)
  PLOTP (LOG OBS)
  PLOTWRITELIB = OUTLIB;
run;
```

This example produces the following output.

Fit Summary

The Fit Summary table records the `PROC GMM` options selected, the number of model components in the final model, and some information about the goodness of fit of the model to the original data.

In this example, you can see that the final model has three components (as specified) and that although a maximum of 100 iterations was specified, the final model reached the specified convergence tolerance after 43 iterations.

The log-likelihood value is the log of the product of the individual likelihoods, and gives a measure of how well the final model fits the training data. The higher the value, the larger the product of the individual likelihoods, and the better the derived Gaussian mixture model fits the data.

The AIC value gives a measure of how well the final model fits the data, compared with the other models that were evaluated in the iterative process, as measured by the *Akaike information criterion*.

The BIC value gives a similar measure of how well the final model fits the data, compared with the other models that were evaluated in the iterative process, but uses the *Bayesian information criterion* as the measure. The two criteria are very similar, but differ in the way they penalise the number of parameters (and hence the number of components) in a model. The metrics used are different, but in both cases, models with fewer parameters are preferred,

As the AIC and BIC values are comparative values, they aren't that helpful for a single model. But the AIC or BIC value for a model can be compared with the value for another model to decide which of the models is likely to be a better fit for the observations in the population.

Fit Summary	
Attribute	Value
Number Of Observations	150
Number Of Predictors	2
Number Of Components	3
Type Of Initialisation	Custom
Number Of Initialisations	1
Warm Start	NO
Type Of Covariance Matrix	FULL
Tied Covariance Matrices	NO
Convergence Tolerance	0.00100
Posterior Probability Tolerance	0.00000
Maximum Number Of Iterations	100
Random seed	12345
Iteration Count	43
Log-Likelihood	-135.47410
AIC	306.94821
BIC	361.13964

Model Parameters

The Model Parameters table shows the model parameters for the final model. In this example, there are eight rows for each component in the model.

- **Weight:** the weight assigned to this component.
- **Means:** the mean value of each of the variables (here, petal length and petal width) for this component.
- **Root Determinant:** the square root of the determinant of the precision matrix.
- **Precision** (four values in this example): the entries in the precision matrix for this component.

In this example, the weights for the three components are 0.33333, 0.34517 and 0.32149, which indicates that the probability distribution for each component contributes about equally to the overall probability distribution predicted by the Gaussian mixture model.

Attribute	Model Parameters			Value
	Component	Row	Column	
Weight	1	1	1	0.33333
Mean	1	1	1	1.46200
Mean	1	2	1	0.24800
RootDeterminant	1	1	1	58.90905
Precision	1	1	1	37.81544
Precision	1	1	2	-20.21066
Precision	1	2	1	-20.21066
Precision	1	2	2	102.57046
Weight	2	1	1	0.34517
Mean	2	1	1	4.29501
Mean	2	2	1	1.33984
RootDeterminant	2	1	1	16.11413
Precision	2	1	1	11.17182
Precision	2	1	2	-21.17582
Precision	2	2	1	-21.17582
Precision	2	2	2	63.38094
Weight	3	1	1	0.32149
Mean	3	1	1	5.56198
Mean	3	2	1	2.03692
RootDeterminant	3	1	1	7.08303
Precision	3	1	1	3.64371
Precision	3	1	2	-2.40393
Precision	3	2	1	-2.40393
Precision	3	2	2	15.35475

Score

The Score table (extracts below) lists each observation in the dataset being scored, and for each, the values of the two variables that contribute to the Gaussian mixture model, the probability density as predicted by the model at that point, and the component that contributes the largest value to the density at that point. For the observations in this dataset, the component with the largest contribution predicts the cluster (the iris species) that the observation is most likely to belong to. In most cases, the species of iris predicted by the model is the same as the actual species.

The score table indicates a clear distinction between observations from the setosa species and observations from the other two species, but there is a less clear distinction between the versicolor and virginica observations. In the next section, you will see that the component density plots show this clearly.

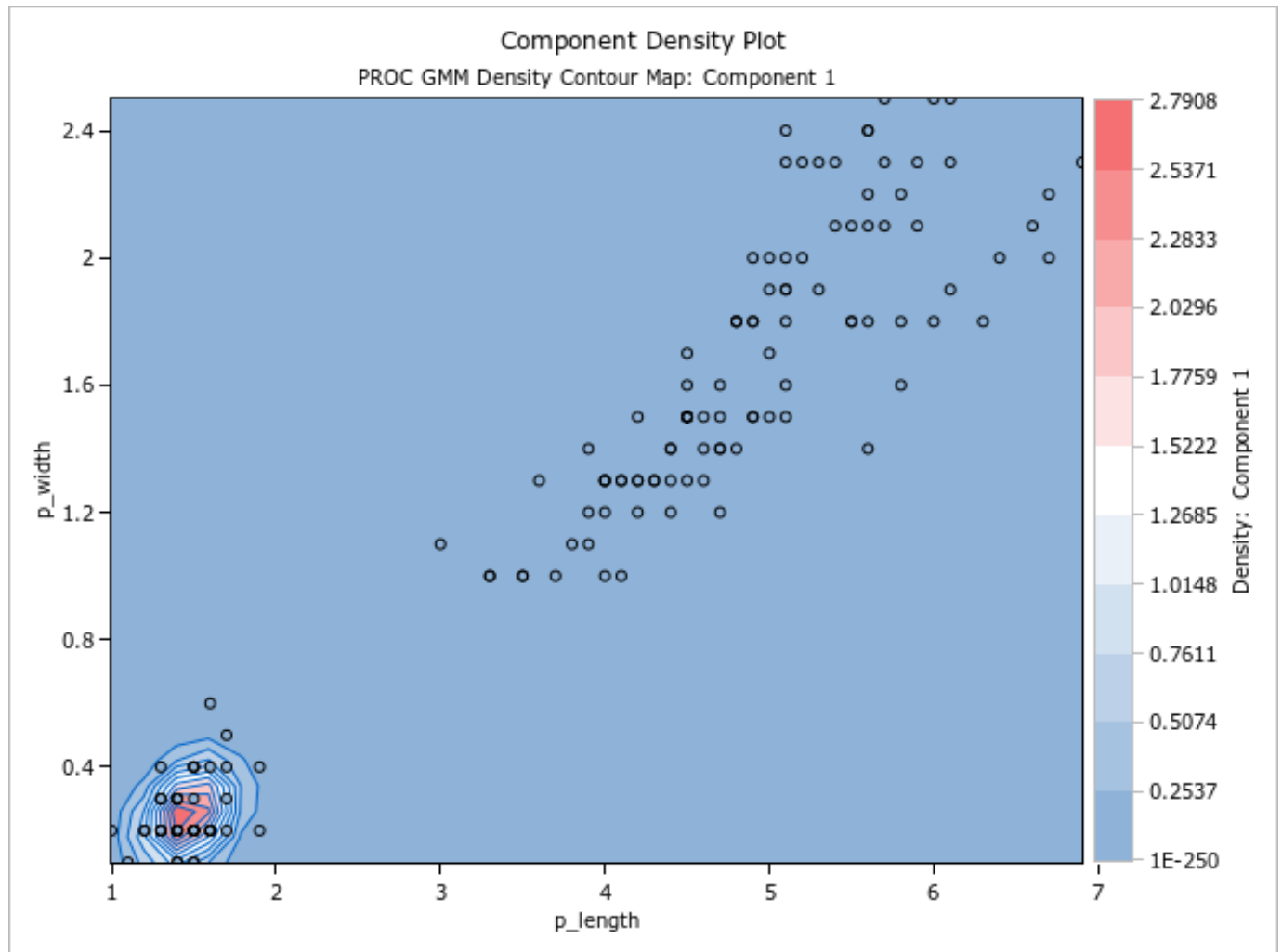
You can see that many of the setosa observations have relatively high density values, and, in each case, component 1 has contributed the most to the density at that point. But many of the versicolor and virginica observations have relatively low density values, and in a few cases, the largest component contribution differs from the other observations from that iris species.

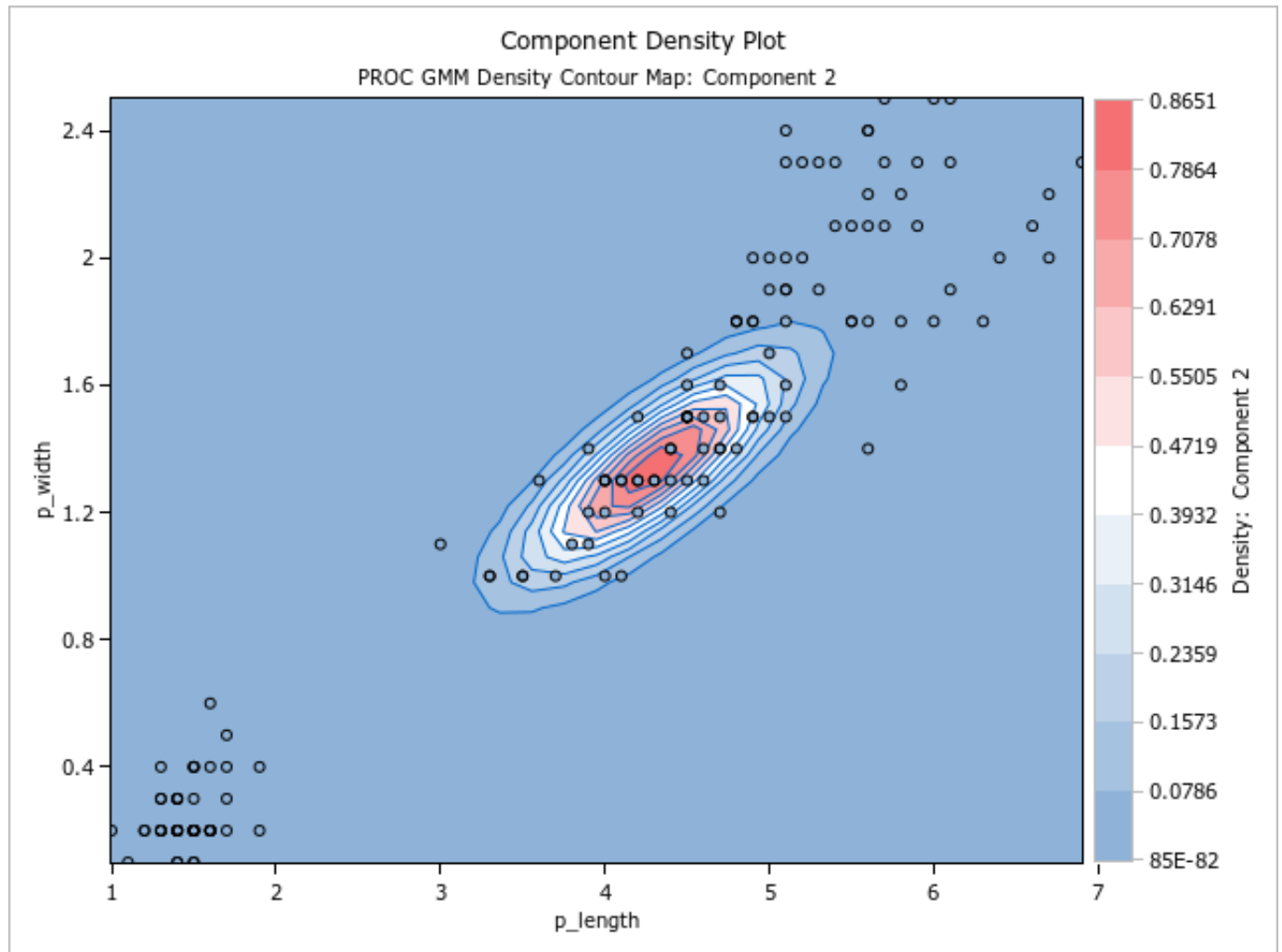
species	p_length	Score		Density
		p_width	Component	

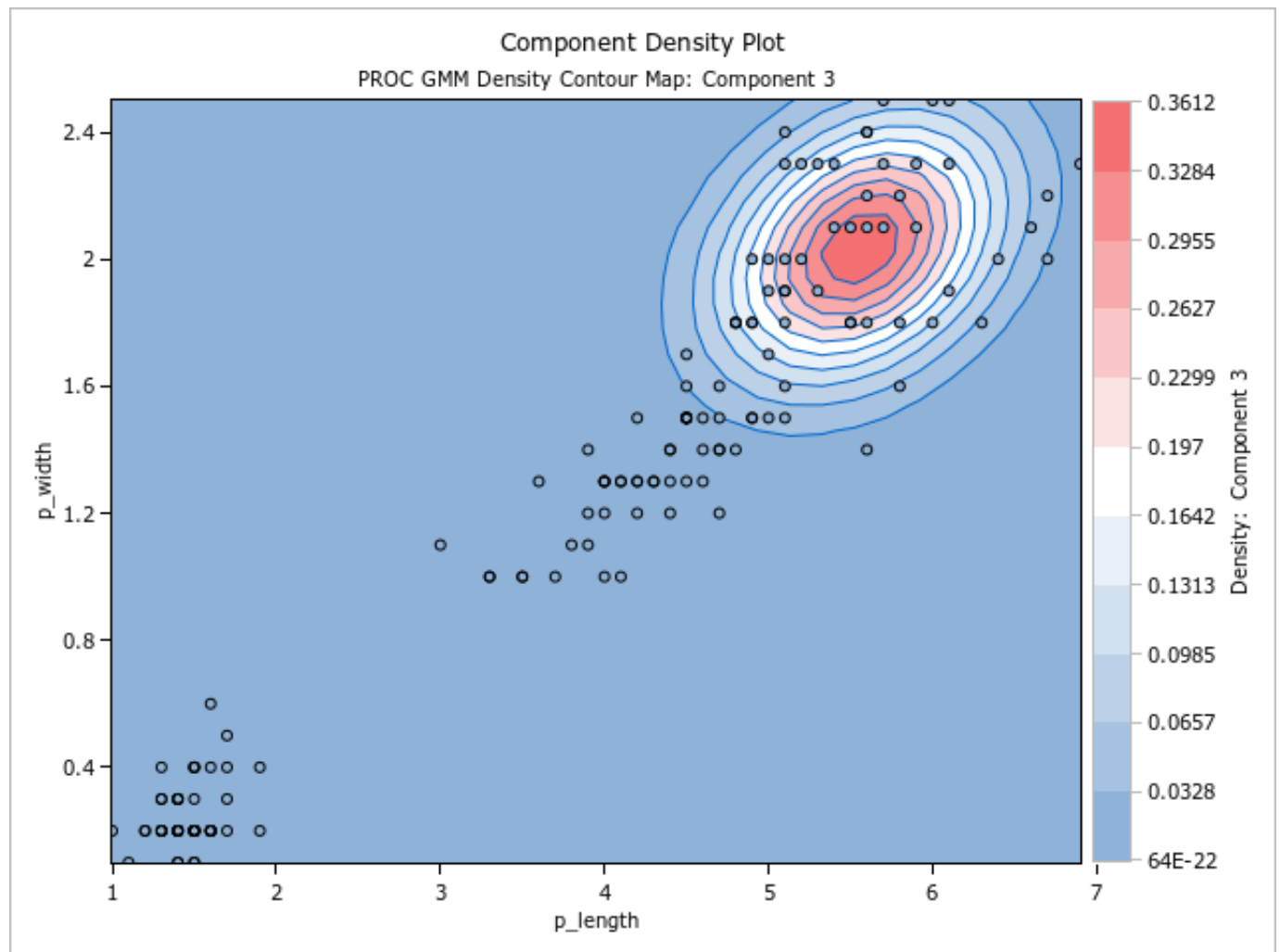
setosa	1.4	0.2	1	2.74234
setosa	1.4	0.2	1	2.74234
setosa	1.3	0.2	1	1.97840
setosa	1.5	0.2	1	2.60433
setosa	1.4	0.3	1	2.37023
setosa	1.7	0.4	1	0.68027
setosa	1.4	0.3	1	2.37023
setosa	1.5	0.2	1	2.60433
setosa	1.4	0.2	1	2.74234
setosa	1.5	0.1	1	0.88266
. . . .				
versicolor	3.6	1.3	2	0.10203
versicolor	4.4	1.4	2	0.85654
versicolor	4.5	1.5	2	0.64233
versicolor	4.1	1	2	0.07502
versicolor	4.5	1.5	2	0.64233
versicolor	3.9	1.1	2	0.44487
versicolor	4.8	1.8	3	0.16180
versicolor	4	1.3	2	0.66506
versicolor	4.9	1.5	2	0.43747
versicolor	4.7	1.2	2	0.05989
. . . .				
virginica	6	2.5	3	0.08020
virginica	5.1	1.9	3	0.26365
virginica	5.9	2.1	3	0.30140
virginica	5.6	1.8	3	0.25643
virginica	5.8	2.2	3	0.29272
virginica	6.6	2.1	3	0.05780
virginica	4.5	1.7	2	0.10072
virginica	6.3	1.8	3	0.05741
virginica	5.8	1.8	3	0.19358
virginica	6.1	2.5	3	0.07504
. . . .				

Component Density Plots

The component density plots are individual contour plots for each component in the mixture model, showing the distribution of the data points and the amount which that component contributes to the model each point.

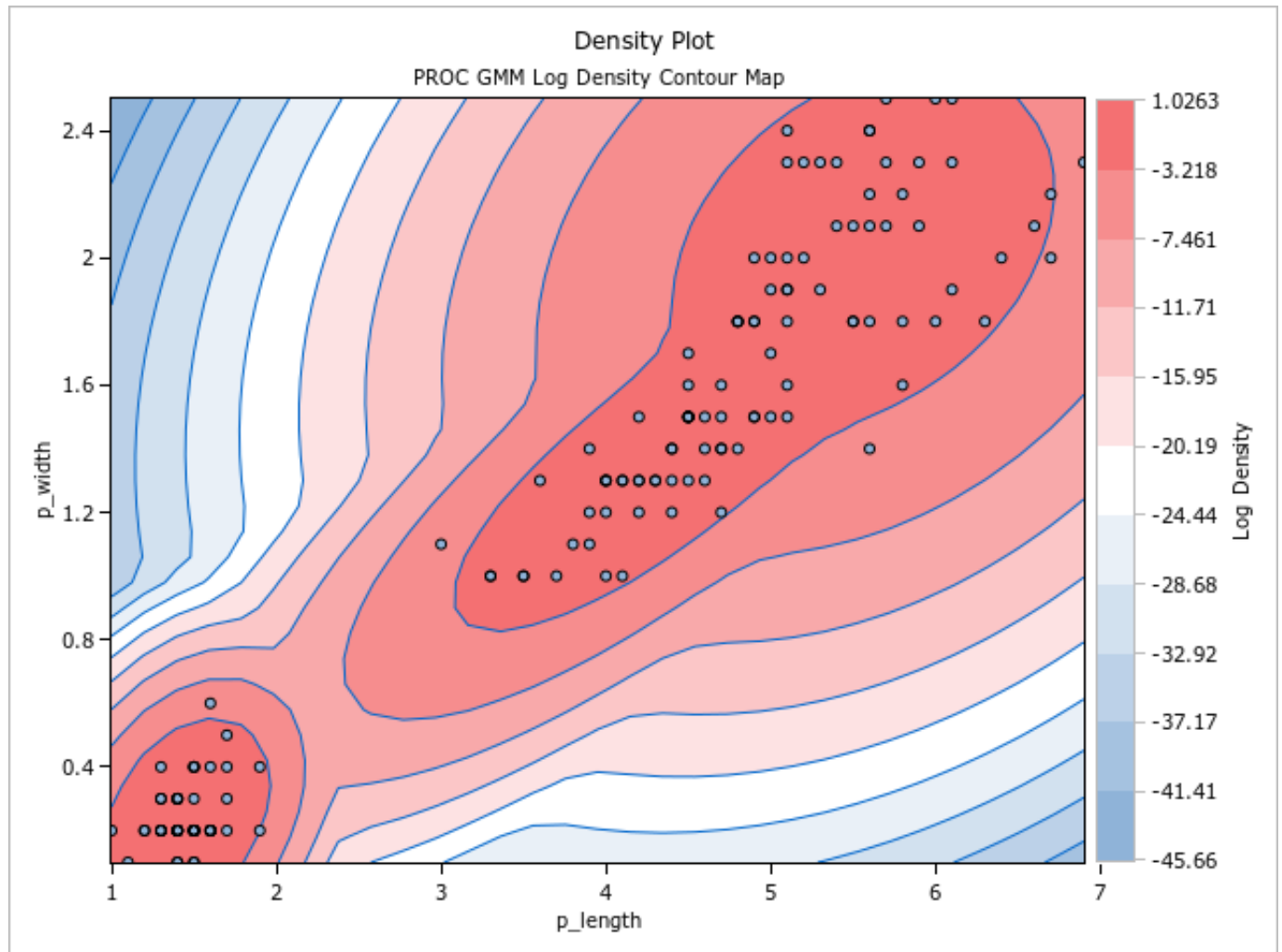






Density Plot

The density plot shows the overall probability density at each point as predicted by the mixture model, and the distribution of the data points. This example shows that the probability density function for the derived model places most of the most of the observations in the higher density areas, as expected.



Example (random initialisation with AIC model selection)

Again, the Iris dataset is supplied as a training dataset to `PROC GMM`, then the derived Gaussian mixture model is used to score the same dataset.

This example uses random initialisation (`INIT=RAND`) with an unspecified number of components to derive the parameters for the Gaussian mixture model. Since the number of components is not known, model selection is used to determine the most suitable number of components.

- models with one, two, three, four and five components are generated (`KMAX=5`)
- the Akaike Information Criterion is used to choose the best model from these five models (`SELECT=AIC`)
- a summary table is produced showing the various models that were evaluated (`SUMMARY`)

As before, the `VAR` statement specifies the predictor variables, `p_length` and `p_width`.

Also as before, the `SCORE` statement uses the Gaussian mixture model to score the same dataset, and the output includes the value of the `species` variable, density plots for each component and for the overall mixture density.

```
PROC GMM
  DATA = MYLIB.iris
  COVARIANCE = FULL (REG = 0.000001)
  INIT = RAND
  SEED = 12345
  SELECT = AIC (KMAX = 5 SUMMARY)
  CTOL = 0.001
  MAXITER = 1000;
VAR p_length p_width;
SCORE DATA = MYLIB.iris0 /
  OUT = OUTLIB.score_init_rand1_select_aic
  PRINT (var = species)
  PLOTK (OBS UNPACK)
  PLOTP (LOG OBS)
  PLOTWRITELIB = OUTLIB;
RUN;
```

This example produces the following output.

Fit Summary

In this example, you can see that the final model has four components, from a maximum of the five that were specified, and that although a maximum of 100 iterations was specified, the final model reached the specified convergence tolerance after 33 iterations.

The log-likelihood is slightly larger than for the model derived using custom initialisation (-123.08735 instead of -135.47410) so this Gaussian mixture model fits the training data slightly better than the custom initialisation model.

The AIC value, at 294.17469, is slightly lower than the value of 306.94821 for the custom initialisation model, so using this criterion to assess the models, this model is the preferred model. The BIC value, at 366.42994, although not used as a criterion for selecting this model, is slightly higher than the custom initialisation value of 361.13964. So using that criterion, the custom initialisation model would be preferred.

Fit Summary	
Attribute	Value
Number Of Observations	150
Number Of Predictors	2
Number Of Components	4
Type Of Initialisation	Random Means
Number Of Initialisations	1
Warm Start	NO
Type Of Covariance Matrix	FULL
Tied Covariance Matrices	NO
Convergence Tolerance	0.00100
Posterior Probability Tolerance	0.00000
Maximum Number Of Iterations	1000
Random seed	12345
Selection Algorithm	Expectation Maximisation

Selection Criterion	AIC
Maximum Number Of Components	5
Iteration Count	33
Log-Likelihood	-123.08735
AIC	294.17469
BIC	366.42994

Model Parameters

The Model Parameters table is similar to the Model Parameters table for the model derived using custom initialisation, but this model has four components.

The parameters for the first component are almost identical to the parameters for the first component of the custom initialisation model. The parameters for the second component are similar for each model. The parameters for the fourth component of this model are reasonably similar to the parameters for the third component of the custom initialisation model. But this model has an extra component, with a weight of 0.06866.

Attribute	Model Parameters			Value
	Component	Row	Column	
Weight	1	1	1	0.33333
Mean	1	1	1	1.46200
Mean	1	2	1	0.24800
RootDeterminant	1	1	1	58.90977
Precision	1	1	1	37.81553
Precision	1	1	2	-20.21055
Precision	1	2	1	-20.21055
Precision	1	2	2	102.57234
Weight	2	1	1	0.33503
Mean	2	1	1	4.30380
Mean	2	2	1	1.33214
RootDeterminant	2	1	1	16.33791
Precision	2	1	1	11.06313
Precision	2	1	2	-23.64359
Precision	2	2	1	-23.64359
Precision	2	2	2	74.65758
Weight	3	1	1	0.06866
Mean	3	1	1	6.29352
Mean	3	2	1	1.90749
RootDeterminant	3	1	1	28.02260
Precision	3	1	1	50.84587
Precision	3	1	2	-77.06190
Precision	3	2	1	-77.06190
Precision	3	2	2	132.23891
Weight	4	1	1	0.26298
Mean	4	1	1	5.31092
Mean	4	2	1	2.05364
RootDeterminant	4	1	1	12.43911
Precision	4	1	1	11.74204
Precision	4	1	2	-13.24572
Precision	4	2	1	-13.24572
Precision	4	2	2	28.11951

Model Summary

The Model Summary table is an optional output when `SELECT=AIC` or `SELECT=BIC`. It shows the parameters for each of the `KMAX` models that were considered. In this case, `SELECT=AIC` was specified, so the model that minimised the AIC value (the model with four components) was selected as the optimal model. If `SELECT=BIC` had been specified, the model with three components would have been selected as the optimal model.

Model Summary				
Number of Components	Number of Free Parameters	Log-Likelihood	AIC	BIC
1	6	-272.86299	557.72598	575.78979
2	12	-154.88986	333.77971	369.90734
3	18	-134.55414	305.10828	359.29972
4	24	-123.08735	294.17469	366.429945
5	30	-125.95451	311.90903	402.22809

Score

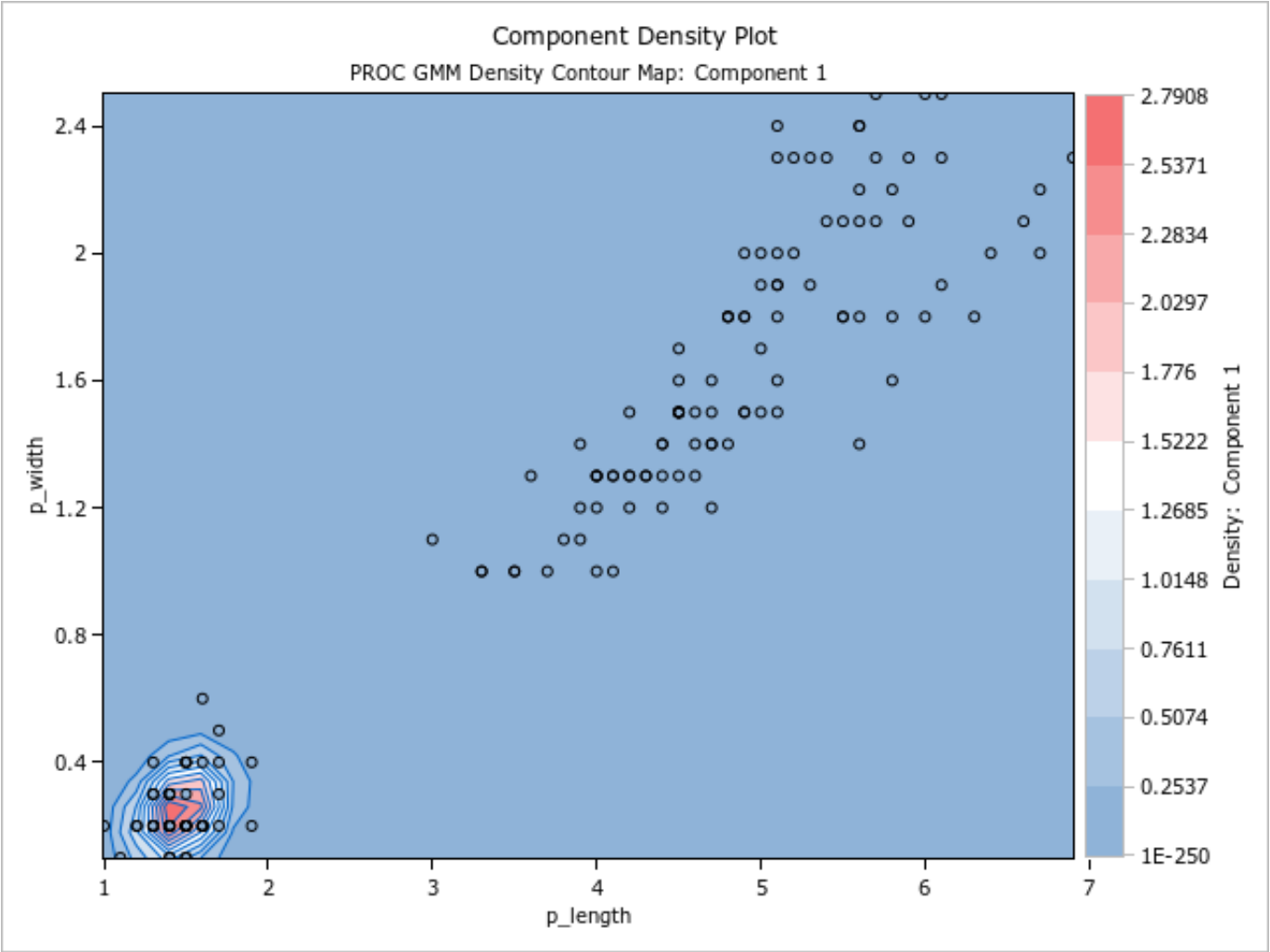
As before, the score table indicates a clear distinction between observations from the *setosa* species and observations from the other two species, but there is a less clear distinction between the *versicolor* and *virginica* observations. In the next section, you will see that the component density plots show this clearly.

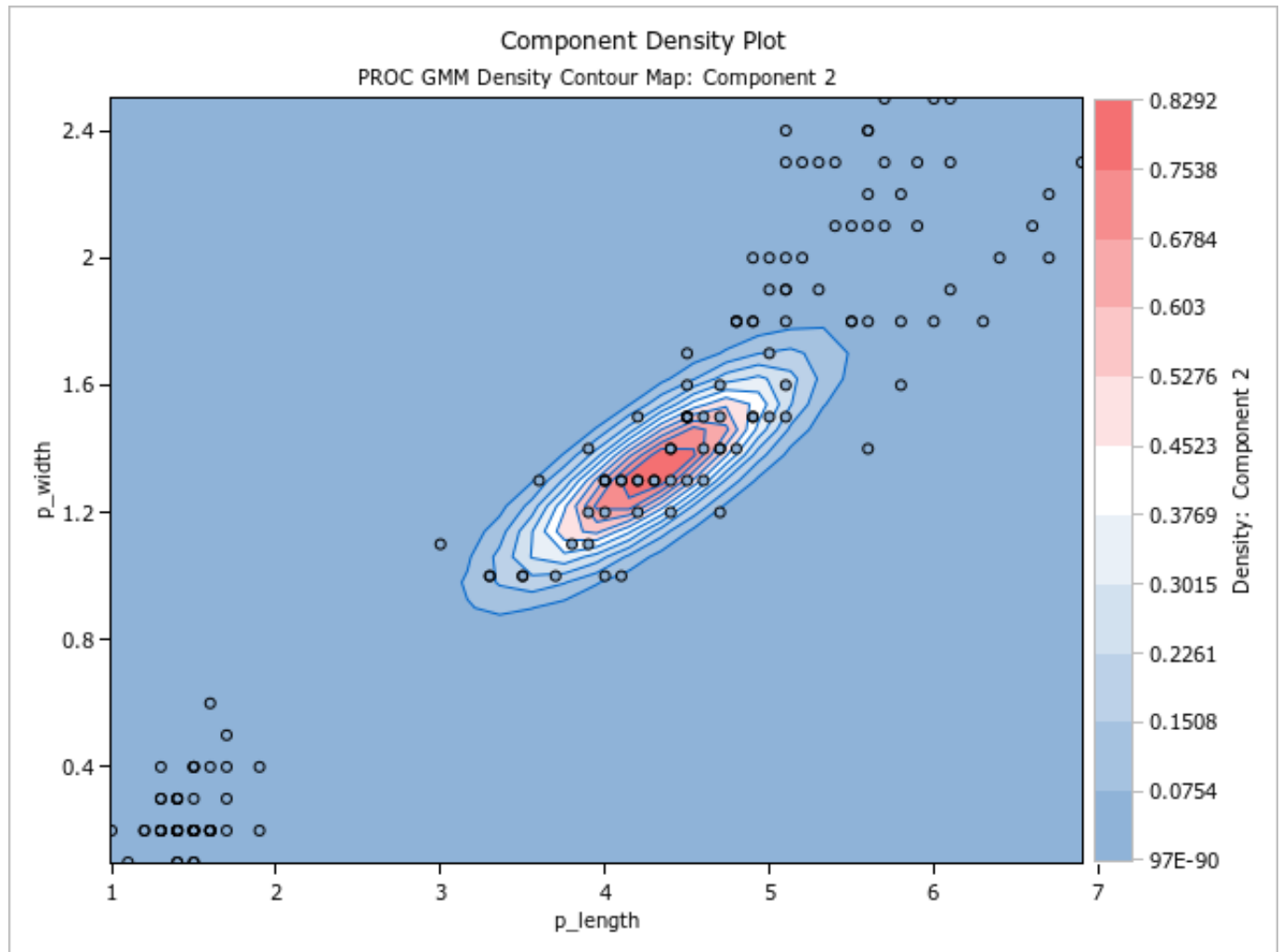
species	p_length	Score p_width	Component	Density
setosa	1.4	0.2	1	2.74237
setosa	1.4	0.2	1	2.74237
setosa	1.3	0.2	1	1.97842
setosa	1.5	0.2	1	2.60436
setosa	1.4	0.3	1	2.37024
setosa	1.7	0.4	1	0.68025
setosa	1.4	0.3	1	2.37024
setosa	1.5	0.2	1	2.60436
setosa	1.4	0.2	1	2.74237
setosa	1.5	0.1	1	0.88266
...				
versicolor	3.6	1.3	2	0.09255
versicolor	4.4	1.4	2	0.83942
versicolor	4.5	1.5	2	0.59213
versicolor	4.1	1	2	0.05619
versicolor	4.5	1.5	2	0.59213
versicolor	3.9	1.1	2	0.43449
versicolor	4.8	1.8	4	0.26854
versicolor	4	1.3	2	0.63725
versicolor	4.9	1.5	2	0.50682
versicolor	4.7	1.2	2	0.05733
...				
virginica	6	2.5	4	0.11445
virginica	5.1	1.9	4	0.44875
virginica	5.9	2.1	4	0.09526

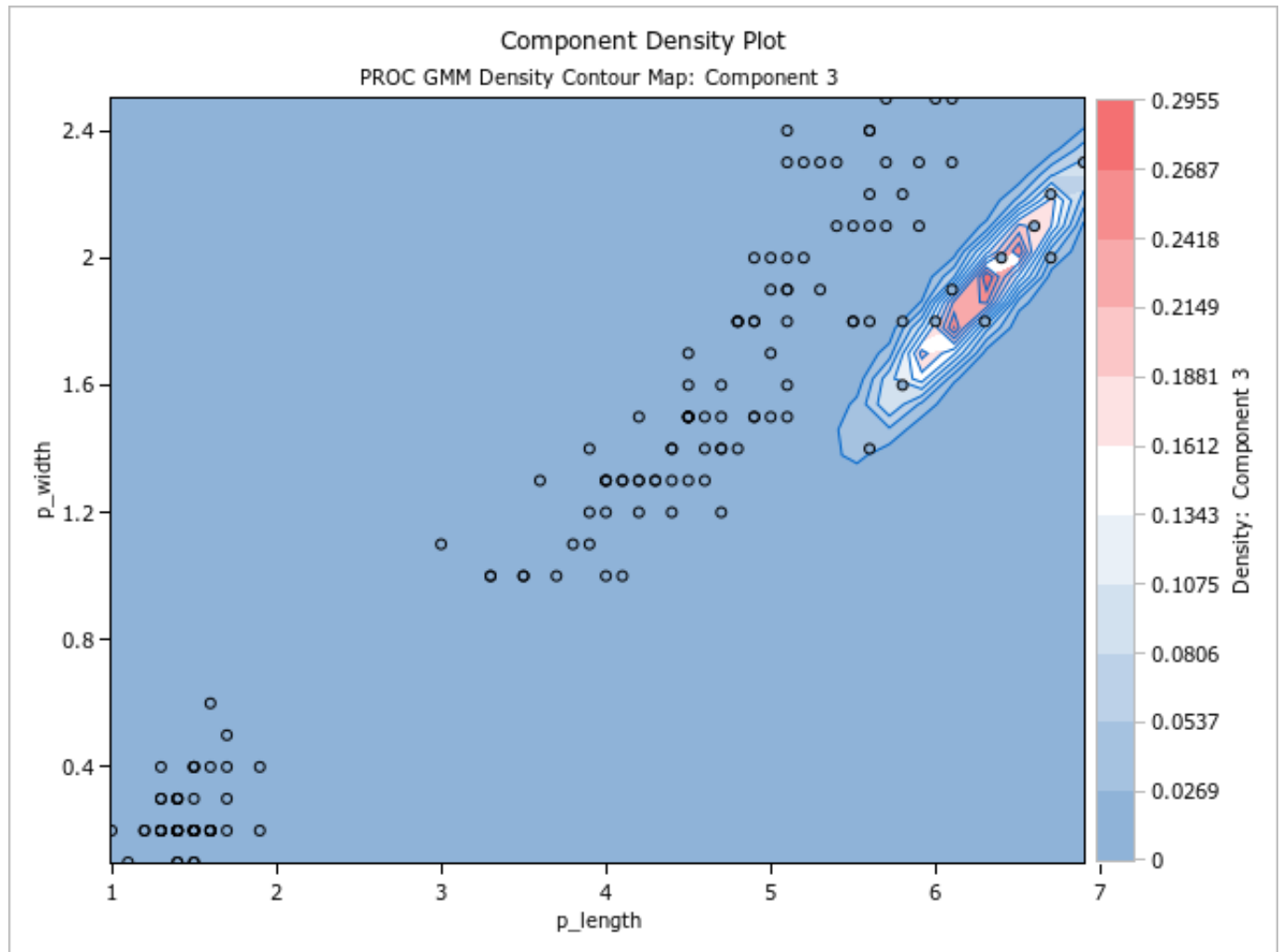
virginica	5.6	1.8	4	0.08729
virginica	5.8	2.2	4	0.24417
virginica	6.6	2.1	3	0.22887
virginica	4.5	1.7	4	0.10913
virginica	6.3	1.8	3	0.13533
virginica	5.8	1.8	3	0.04330
virginica	6.1	2.5	4	0.08680

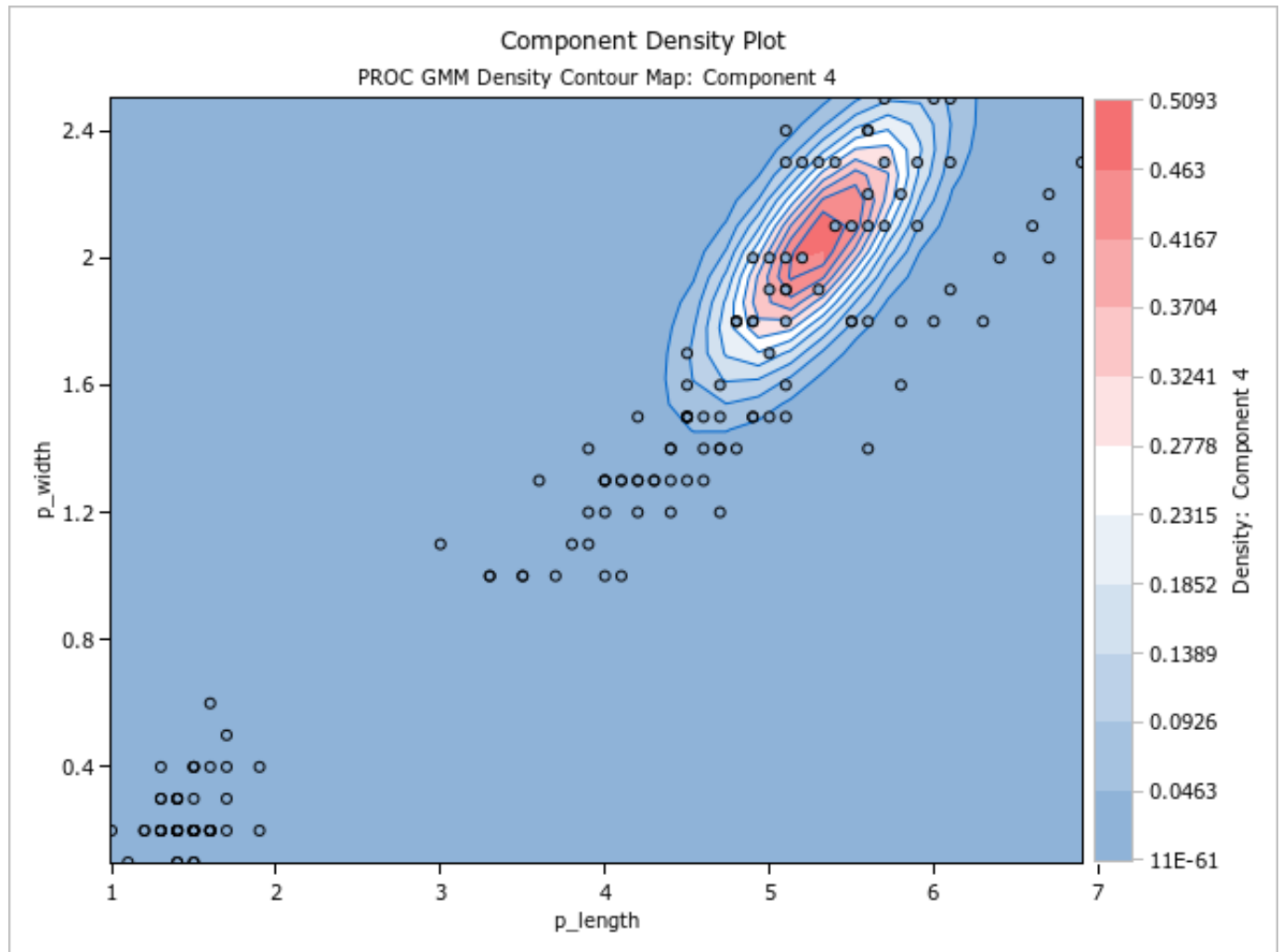
Component Density Plots

The component density plots for the first and second components are similar to the corresponding plots for the custom initialisation model. But the other two plots show differences in the way the model has been calculated to fit the data.



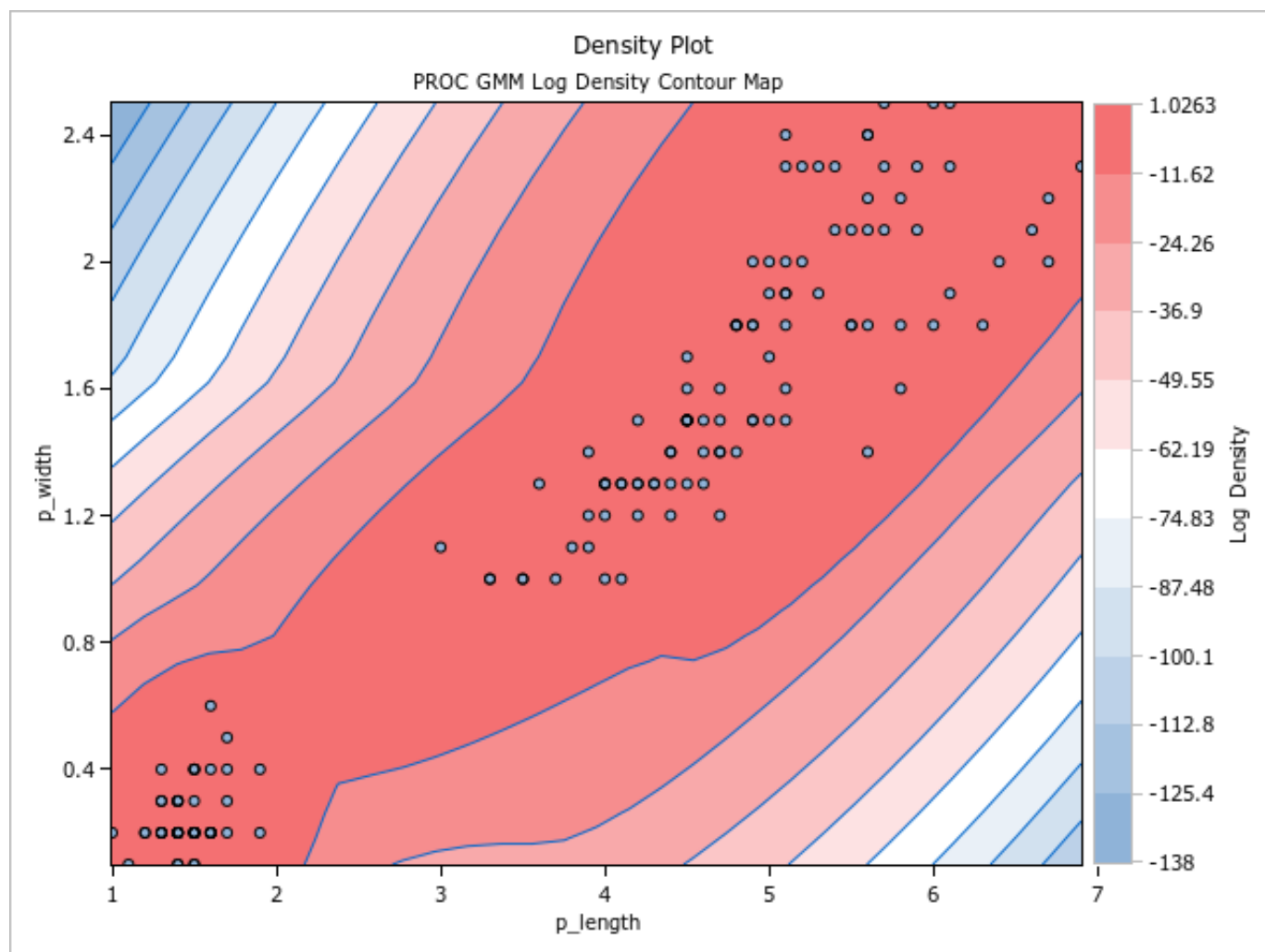






Density Plot

The overall density plot is also slightly different from the custom initialisation model.



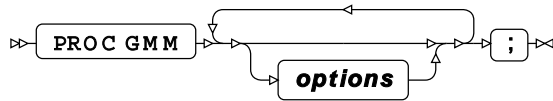
GMM procedure reference

Describes the syntax and options for `PROC GMM` and its contained statements.

<code>PROC GMM</code> ↗	3238
Creates a Gaussian mixture model from the input dataset.	
<code>BY</code> ↗	3246
Groups the observations in the input dataset using one or more specified variables.	
<code>CODE</code> ↗	3246
Outputs the Gaussian mixture model as a file containing data step code, which can subsequently be used to score another dataset using this model.	
<code>SCORE</code> ↗	3247
Uses the current GMM model to score the data in the specified dataset.	
<code>VAR</code> ↗	3249
Specifies the predictor variables for the Gaussian mixture model.	

PROC GMM

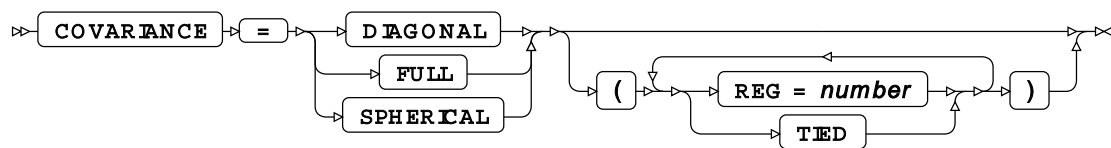
Creates a Gaussian mixture model from the input dataset.



Options

The following *options* are available:

COVARIANCE



Specifies the type of covariance matrices that the model uses. The covariance matrix is symmetric about the leading diagonal. The leading diagonal contains the variance of each of the independent variables that are being modelled, and the other positions contain the covariance of each pair of variables.

DIAGONAL

The off-diagonal elements of the covariance matrices are equal to zero. A diagonal covariance matrix can be represented as a d -dimensional vector, where d is the number of predictor variables specified in the VAR statement.

FULL

The covariance matrices are symmetric about the leading diagonal, and any value may be zero or non-zero. A full covariance matrix is a d by d matrix, where d is the number of predictor variables specified in the VAR statement.

This is the default value.

SPHERICAL

The covariance matrices are the product of a scalar and the identity matrix. A spherical covariance matrix can be represented as a scalar.

Covariance sub-options

The following sub-options are available for any of the COVARIANCE options above.

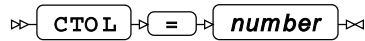
REG

Specifies a regularisation value to add to the diagonal elements of the covariance matrices to try to ensure that they are positive definite. If any covariance matrix is not positive definite, the algorithm will report an error and terminate. In that case, you should try specifying a small positive value for REG, for example, 1E-10.

The default value is 0 (zero).

TIED

Specifies that all components in the Gaussian mixture model use the same covariance matrix.

CTOL

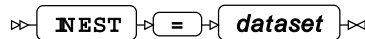
Specifies the tolerance to use when checking for algorithm convergence after each iteration. The algorithm terminates when the change in the value being tested is less than the specified tolerance. If `PROC GMM` specifies an explicit `K` value, or if `K` is not specified and `SELECT = AIC` or `SELECT = BIC`, then the value of the log likelihood is used to determine convergence. If `K` is not specified and `SELECT = VB`, the value of the variational lower bound is used to determine convergence.

If specified, the tolerance must be a positive real number. The default value is 0.001.

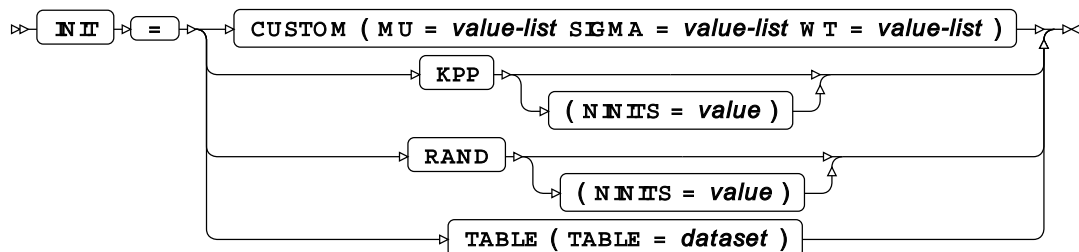
DATA

Specifies the training dataset used by `PROC GMM` to construct the Gaussian mixture model.

If a training dataset is not specified, the most recently-created dataset is used.

INEST

Specifies the name of a dataset that was created by the `OUTEST` option in a previous call of `PROC GMM` and which contains saved model parameters. `PROC GMM` uses the parameters saved in this dataset as the initial values for the model parameters. If either or both of the `INIT` or `SELECT` options were specified when the dataset was created, the same options and values must be specified this time.

INIT

Specifies the type of initialisation to use for the components of the model.

CUSTOM

Specifies that the model uses custom initial values. In this case, values for `MU`, `SIGMA` and `WT` must be supplied, and the `K` option must also be specified.

The following sub-options must be specified with the `CUSTOM` option:

MU

Specifies values for the initial means of the components. There should be dK entries in *value-list*, where d is the number of predictor variables specified in the `VAR` statement and K is the value specified for `K`.

For example, if you wanted to specify the initial means for a dataset with two predictor variables and three components, you need to provide six values. These correspond to the two component means of the predictor variables for the first component in the mixture model, followed by the two component means for the second component, followed by the two component means for the third component.

SIGMA

Specifies the initial covariance matrices of the components. The number of entries in *value-list* depends on the `COVARIANCE` option selected:

- If `COVARIANCE=FULL`, *value-list* contains d^2K entries (one d by d matrix for each component) where d is the number of predictor variables specified in the `VAR` statement and K is the value specified for `K`.
- If `COVARIANCE=FULL (TIED)`, each of the K covariance matrices are the same, so *value-list* contains d^2 entries, where d is the number of predictor variables specified in the `VAR` statement.
- If `COVARIANCE=DIAGONAL`, only the diagonal entries are required, so *value-list* contains dK entries, where d is the number of predictor variables specified in the `VAR` statement and K is the value specified for `K`.
- If `COVARIANCE=DIAGONAL (TIED)`, each of the K covariance matrices are the same, so *value-list* contains one entry for each predictor variable in the `VAR` statement.
- If `COVARIANCE=SPHERICAL`, each diagonal entry is the same, so *value-list* contains K entries, where K is the value specified for `K`.
- If `COVARIANCE=SPHERICAL (TIED)`, each of the K covariance matrices are the same, so *value-list* contains one entry.

WT

Specifies values for the initial mixture weights of the components. There should be K entries in *value-list*, where K is the value specified for `K`.

KPP

Specifies that the *k-means++* algorithm is used to derive the initial component means. The initial mixing proportions are uniform and the initial covariance matrices are derived from the covariances of the dataset.

If `INIT = KPP`, `K` is optional. If `K` is specified, the algorithm uses the specified value for the number of components. If `K` is not specified then the values in the `SELECT` option are used to determine the optimal number of components and the initial values.

The following sub-options can optionally be specified:

NINITs

Specifies the number of times the algorithm is run with different initial values. If specified, must be a positive integer. The default value is 1.

RAND

Specifies that a number of samples are selected at random for the initial component means, where `K` is the value specified for `K`. The initial mixing proportions are uniform and the initial covariance matrices are derived from the covariance of the dataset.

If `INIT = RAND`, `K` is optional. If `K` is specified, then `K` samples are selected initially, where `K` is the value specified for `K`. If `K` is not specified, the values in the `SELECT` option are used to determine the optimal number of components and the initial values.

This is the default value.

The following sub-options can optionally be specified:

NINITs

Specifies the number of times the algorithm is run with different initial values. If specified, must be a positive integer. The default value is 1.

TABLE

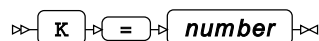
Specifies that a dataset table is used to initialise the components. In this case, the `K` option must also be specified, and must match the value derived from the table in the dataset.

This option is useful if you have already completed some iterations of a complex model which terminated before convergence, and want to continue from where you left off. You can supply the contents of the Model Parameters table from the previous run, and the calculations will continue starting with the results from the final iteration of the previous run.

dataset

Specifies the dataset containing the table to be used to initialise the components. This dataset must have the same structure as the Model Parameters table that is output when `PROC GMM` runs. It must also contain the number of components specified by `K`.

K



A positive integer that specifies the number of components in the Gaussian mixture model.

This option is required if `INIT=CUSTOM` or `INIT=TABLE` is specified. If `INIT = KPP` or `INIT = RAND`, it is optional.

If K is specified, Expectation Maximisation is used to determine the other model parameters. If K is not specified, the model selection process is used to determine the correct number of components and their corresponding parameters. You can use the `PROC GMM SELECT` option to control the model selection process

MAXITER

➤ `MAXITER` ➤ `=` ➤ `number` ➤

Specifies the maximum number of iterations allowed when creating a Gaussian mixture model. If specified, must be a positive integer. The default value is 100.

OUTEST

➤ `OUTEST` ➤ `=` ➤ `dataset` ➤

Specifies that the final model parameters are to be saved in a dataset. The parameters saved in this dataset can later be used as the initial parameters for a new model.

PPTOL

➤ `PPTOL` ➤ `=` ➤ `number` ➤

Specifies the value below which posterior probabilities are clipped to 0 (zero) to try to speed up convergence. If specified, must be a positive real number. The default value is 0.

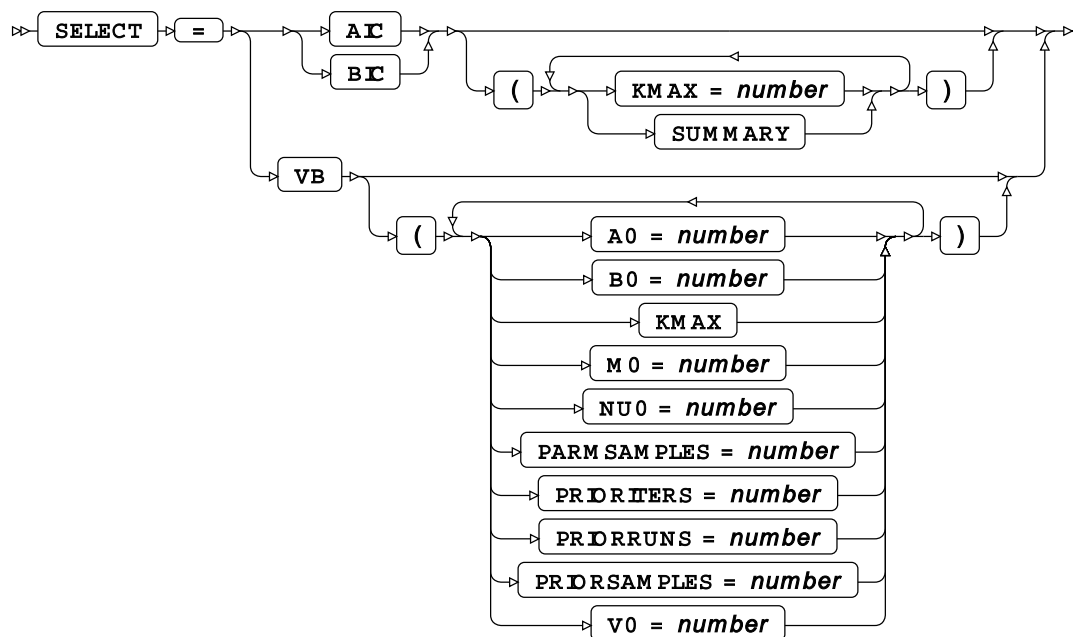
This value is ignored unless `PROC GMM` also specifies K , `SELECT=AIC` or `SELECT=BIC`.

SEED

➤ `SEED` ➤ `=` ➤ `number` ➤

Specifies the seed to be used by the various random number generators in the GMM implementation. If specified, must be a non-negative integer. The default value is 12345.

Note that the specified seed is not just to derive the initial values of components when `INIT=RAND` is selected: it is used throughout the GMM implementation. The chosen seed value may have a considerable effect on the model that is derived, as the model may converge to different local maxima for different choices of the starting point and the incremental iteration parameters.

SELECT

Specifies the model selection criteria that are used to select the most appropriate model from the models being evaluated. The `SELECT` option is only required if the number of components in the model is not already known. This option is ignored if `PROC GMM` also specifies an explicit value for `K`.

If you know the maximum number of components in the mixture (`KMAX`), you can choose any of the model selection criteria, `AIC`, `BIC` or `VB`. If you do not know the value of `KMAX`, you must use `VB`.

If you choose `VB` as the model selection criteria and also know `KMAX`, then you must specify the prior hyperparameters `A0`, `B0`, `M0`, `NU0` and `V0`, or accept the default values. If you do not know `KMAX` either, instead of the prior hyperparameters, you must also specify `PRIORITERS`, `PRIORRUNS` and `PRIORSAMPLES`, or accept the default values, so that the prior hyperparameters can also be estimated.

AIC

Specifies that the *Akaike Information Criterion* metric is used to evaluate models. In this case, `KMAX` models are created, one with a single component, one with two components, and so on, up until one with `KMAX` components. For each model, the initial model parameters are calculated using the specified option for creating initial values (`INIT=KPP` or `INIT=RAND`), then the parameters are refined using expectation maximisation (EM). Finally, the Akaike Information Criterion is used to select the optimal model from the `KMAX` models that were evaluated.

KMAX

Specifies the maximum number of components to use when performing model selection.

If specified, must be a positive integer. The default value is 20.

SUMMARY

Specifies that a Model Summary table is output, showing the metrics for each model that was evaluated.

BIC

Specifies that a *Bayesian Information Criterion* metric is used to evaluate models. This is the default model selection method. In this case, `KMAX` models are created, one with a single component, one with two components, and so on, up until one with `KMAX` components. For each model, the initial model parameters are calculated using the specified option for creating initial values (`INIT=KPP` or `INIT=RAND`), then the parameters are refined using expectation maximisation (EM). Finally, the Bayesian Information Criterion is used to select the optimal model from the `KMAX` models that were evaluated.

KMAX

Specifies the maximum number of components to use when performing model selection. If specified, must be a positive integer. When `SELECT=BIC`, the default value is 20.

SUMMARY

Specifies that the metrics for each model that was evaluated are displayed in the ODS Model Summary table.

VB

Specifies that a variational Bayesian (approximate inference) approach is used to find the most appropriate model.

A0

Specifies the prior hyperparameter corresponding to the concentration weight. If specified, must be a positive real number. The default value is 1. The specified value is only used if `KMAX` is also supplied.

B0

Specifies the prior hyperparameter corresponding to the mean precision. If specified, must be a positive real number. The default value is 1. The specified value is only used if `KMAX` is supplied.

KMAX

Specifies the maximum number of components to use when performing model selection. If specified, must be a positive integer.

If specified, the `PRIORITERS`, `PRIORRUNS` and `PRIORSAMPLES` (if also specified) are ignored.

If not specified, the procedure determines the maximum number of components during the prior processing stage. In this case, `PRIORITERS`, `PRIORRUNS` and `PRIORSAMPLES` (or their default values) are used to determine the maximum number of components.

M0

Specifies the prior hyperparameter corresponding to the means. This is a list of d real numbers where d is the number of predictor variables specified in the `VAR` statement. If not specified, the default value for each entry is 0 (zero). The specified value is only used if `KMAX` is also supplied.

NU0

Specifies the prior hyperparameter corresponding to the degrees of freedom. If specified, must be a positive integer, greater than or equal to $d - 1$, where d is the number of predictor variables specified in the `VAR` statement. If not specified, the default value is d . The specified value is only used if `KMAX` is also supplied.

PARMSAMPLES

Specifies the number of samples to use when stochastically updating the component parameters. If specified, must be a non-negative integer. If this value is 0, the means of the prior distributions are used to update the component parameters. If not specified, the default value is 0 (zero).

PRIORITERS

Specifies the number of iterations to use when determining the maximum number of components and the prior hyperparameters. If specified, must be a positive integer. If not specified, the default value is 100. This sub-option is ignored if the `KMAX` sub-option has also been specified.

PRIORRUNS

Specifies the number of runs to use when determining the maximum number of components and the prior hyperparameters. If specified, must be a positive integer. If not specified, the default value is 1. This sub-option is ignored if the `KMAX` sub-option has also been specified.

PRIORSAMPLES

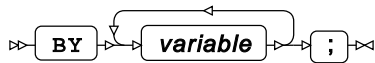
Specifies the number of samples to use when stochastically updating the prior hyperparameters. If specified, must be a positive integer. If not specified, the default value is 1. This sub-option is ignored if the `KMAX` sub-option has also been specified.

V0

Specifies the scale prior hyperparameter. This is a d by d matrix where d is the number of predictor variables in the `VAR` statement. The default is the identity matrix. If `COVARIANCE=FULL`, then *value-list* contains d^2 entries; if `COVARIANCE=DIAGONAL`, then *value-list* contains d entries (all the non-diagonal entries are zero) and if `COVARIANCE=SPHERICAL`, then *value-list* contains a single scalar (all the non-diagonal entries are zero, and all the diagonal entries have the same value). The specified value is only used if `KMAX` is also supplied.

BY

Groups the observations in the input dataset using one or more specified variables.

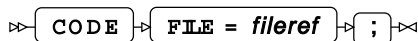


The specified variable or variables are used to separate the input data into groups. `PROC GMM` generates a separate model from the data in each group.

If the `BY` statement is included, the input dataset must be pre-sorted on the specified variable or variables. If a variable is specified as a predictor variable in the `VAR` statement, it cannot also be specified in the `BY` statement.

CODE

Outputs the Gaussian mixture model as a file containing data step code, which can subsequently be used to score another dataset using this model.



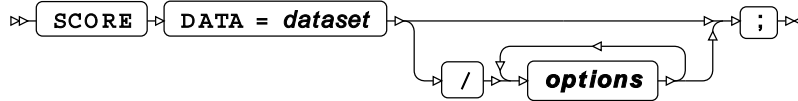
When the code file has been produced, you can write a data step that contains a `SET` statement to specify the dataset to be scored, followed by a `%INCLUDE` statement to include the mixture model code file. The output dataset contains the scored data.

FILE

Specifies the name of the file to contain the data step code statements.

SCORE

Uses the current GMM model to score the data in the specified dataset.



The **SCORE** statement takes the data in the specified dataset and scores it using the GMM model defined in **PROC GMM**. The score results are saved in a table which can be printed or saved in an output dataset.

For each observation in the dataset being scored, the score results table shows the component that the observation is most likely to belong to, and the overall probability density of the distribution at that point. The score results table includes all the data in the input dataset, including observations with missing values. But observations with missing values for predictor variables are not scored.

You can specify multiple **SCORE** statements if required.

You can use the **PRINT** option to print the score results. You can also use **ODS OUTPUT** to save the score results in an output dataset. If the model has one or two predictor variables, you can also produce score plots for the mixture model and its components. Models with one predictor variable produce line plots, and models with two predictor variables produce contour plots.

DATA

Specifies the dataset to score. All the predictor variables specified in the **VAR** statement must be present in the dataset.

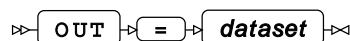
If **PROC GMM** includes a **BY** statement, the dataset to be scored must also contain all the variables mentioned in the **BY** statement, and must be sorted in the order of those variables.

The **DATA** option is mandatory.

Options

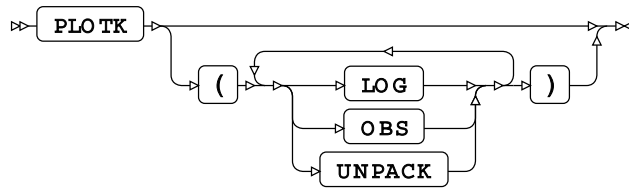
The following *options* are available:

OUT



Specifies the dataset to which the score results are output. The dataset contains the original data, and additional fields showing which component each observation is most likely to have come from and the estimated probability density at that point.. If not specified, no output dataset is produced.

If **PROC GMM** specifies more than one output dataset for score results (for example, if there is more than one **SCORE** statement) then each output dataset must have a unique name.

PLOTK

Specifies that a density plot is produced for each component from the score dataset.

This option is only valid for models that contain one or two predictor variables.

LOG

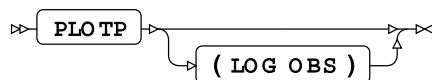
Specifies that the component plots use a natural logarithmic scale for the component densities.

OBS

Specifies that observations are overlaid on top of the plot.

UNPACK

Specifies that the plots are displayed individually. By default, the plots are displayed in a panel.

PLOTP

Specifies that a mixture density plot is produced for the score dataset.

This option is only valid for models that contain one or two predictor variables.

LOG

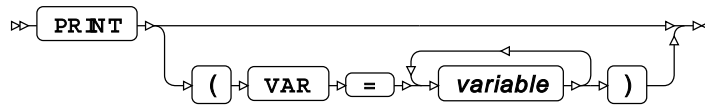
Specifies that the mixture density plots use a natural logarithmic scale for the component densities.

OBS

Specifies that observations are overlaid on top of the plot.

PLOTWRITELIB

Specifies a library where the plot datasets are saved.

PRINT

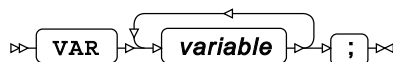
Specifies that the score table for the score dataset is written to ODS output. By default, each entry in the score table contains the predictor variables, which component each observation is most likely to have come from and the estimated probability density at that point. Optionally, you can also include additional variables from the dataset.

VAR

Specifies additional variables from the score dataset to be included in the score table.

VAR

Specifies the predictor variables for the Gaussian mixture model.

**variable**

Specifies one or more predictor variables to which the specified model options apply. Observations with missing values for any of the predictor variables are ignored.

GMM bibliography

These items are referenced in the GMM procedure section.

- [1] Akaike, H., 1973. Information theory and an extension of the maximum likelihood principle. In: B. N. Petrov & F. Csaki, eds. *2nd International Symposium on Information Theory*. Tsahkadsor, Armenia, USSR: Budapest: Akademiai Klado, pp. 267–281.
- [2] Arthur, D. & Vassilvitskii, S., 2007. k-means++: the advantages of careful seeding. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, pp. 1027–1035.
- [3] Bishop, C. M., 2006. *Pattern recognition and machine learning*. 1st ed. New York, USA: Springer.
- [4] Dempster, A. P., Laird, N. M. & Rubin, D. B., 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1), pp. 1–38.
- [5] Rasmussen, C. E., 2000. The Infinite Gaussian Mixture Model. *Advances in Neural Information Processing Systems*, 12(1), pp. 554–560.

- [6] Schwarz, G. E., 1978. Estimating the dimension of a model. *Annals of Statistics*, 6(2), pp. 461–464.
- [7] Wood, F. & Black, M. J., 2012. A nonparametric Bayesian alternative to spike sorting. *Journal of Neuroscientific Methods*, 173(1), pp. 1–12.
- [8] Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), pp. 179–188.

MLP procedure

The MLP procedure enables you to build a multilayer perceptron (MLP) from an input dataset. You can then use the MLP model to analyse other datasets.

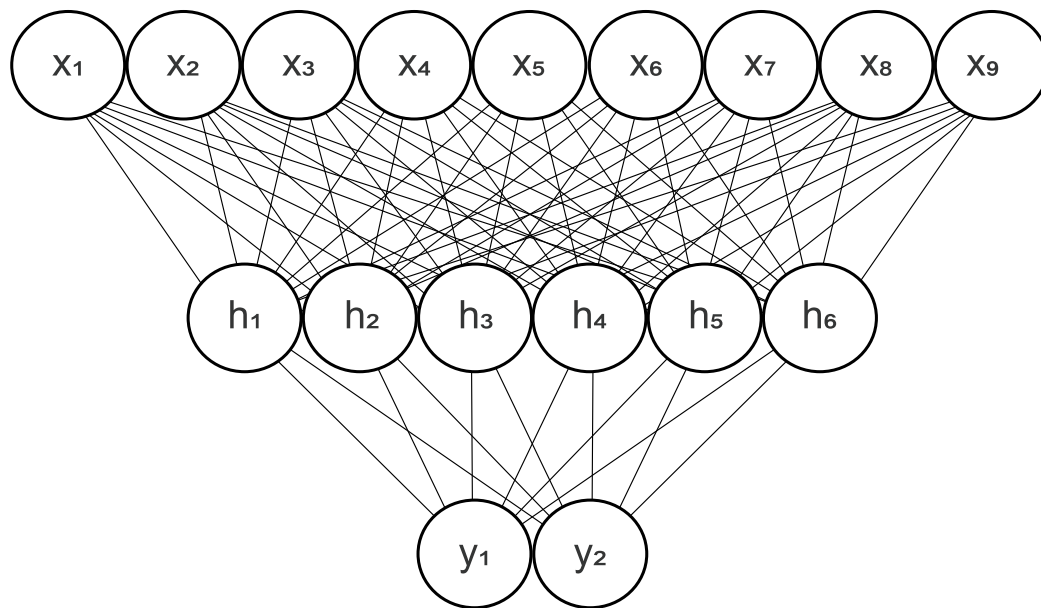
About multilayer perceptrons

Multilayer perceptron (MLP) neural networks are a powerful class of non-linear machine learning algorithms that can be used for classification and regression.

MLPs are complex, non-linear functions, usually conceptualised as layers of simple non-linear functions connected by layers of weights. The layers of non-linear functions are analogous to layers of neurons in biological neural networks, and the weights are analogous to synapses.

Training algorithms can be used to find the weights that enable an MLP to approximate the relationships between the effect variables and a response variable in a set of training data, thereby making it possible to create a neural network model that can predict the values of unknown response variables from known effect variables.

The following is a typical fully-connected MLP with nine input neurons, a single hidden layer containing six neurons, and an output layer containing two neurons:



Every neuron in each layer is connected to every neuron in the preceding layer to produce a fully-connected network. Information about the values of the effect variables in an observation is encoded on the activities of the input neurons, which excite or inhibit the hidden neurons depending on the values of the weights between them.

The hidden neurons then excite or inhibit the output neurons depending on the values of the weights between them. The activities of the output neurons represent the network's prediction of the value of the encoded response variable.

In `PROC MLP`, there is one input neuron for each continuous effect variable. For a categorical effect variable, the number of input neurons on the encoding used for the variable.

For a categorical response variable, there is one output neuron for each value of the response variable. For a given set of input values on the input neurons, the value of each output neuron is an estimate of the probability of the response variable having that value.

For a continuous response variable, there is a single output neuron. For a given set of input values on the input neurons, the value of the output neuron is an estimate of the value of the response variable. The way that this estimate is calculated depends on the selected error function. For example, if `ERROR=SUMOFSQUARES` is specified in the `MODEL` statement, the network output is an estimate of the conditional mean of the Gaussian distribution of the response variable, as a function of the effect variables.

Specifying network structure

The network structure is determined by the number of neurons in the input and output layers, the number of hidden layers, the number of neurons in each hidden layer, and the way each neuron interacts with the neurons in the next layer.

In an MLP, the number of neurons in the input layer is determined automatically by the number, types and encodings of the effect variables specified using the `MODEL` statement. The number of neurons in the output layer is determined by the type and encoding of the response variable. In general, a non-class variable will be represented by a single neuron and a class variable will be represented by one neuron for each class level when it is GLM encoded and by one neuron for each non-reference level for all other encodings.

The number of hidden layers and the number of neurons in each layer determines the range of functions that a neural network can learn. In general, networks with more hidden neurons and more hidden layers can learn more complex functions. These networks will, however, require more training data and careful regularisation to avoid overfitting.

The MLP procedure supports the specification of networks with multiple hidden layers each with arbitrarily many neurons. The practical limit to the size of a network created using the MLP procedure is limited only by a computer's memory. A network defined with `PROC MLP` requires the specification of at least one hidden layer. Alternatively, the functional equivalent of an MLP without a hidden layer can be created using one of the `GENMOD`, `LOGISTIC`, `PROBIT` or `GLM` procedures.

Each neuron performs a non-linear transformation that is referred to as its *activation function*. The `MODEL` statement in the procedure supports a wide variety of different types of activation functions (for more information see [Activation functions](#) (page 3293)).

Input neurons represent the values of effect variables and therefore normally perform no transformations. The MLP procedure enables the specification of a different activation function for the neurons in each hidden layer. This can be specified using the `HIDDEN` option of the `MODEL` statement.

The activation function of output layer neurons is automatically set to `SOFTMAX` if the response variable is a class variable and `LINEAR` otherwise. Output neuron activation functions can be changed using the `OUTPUT` option on the `MODEL` statement.

Preprocessing data

Neural networks typically require effect variables to be standardized in some way.

Standardisation can improve learning performance and help prevent effects with greater variability from having undue influence on the trained network.

By default, `PROC MLP` standardizes the values of effect variables to have zero mean and unit variance. You can use the `PREPROCESS` option of the `MODEL` statement to override this behaviour.

Training a network

The process of training a neural network consists of using a training algorithm to iteratively change the network's weights to minimize a measure of the error between the actual responses in a set of training data and the responses predicted by the network.

The dataset containing the training data is specified by the `DATA` option of `PROC MLP` and must be supplied for the procedure to run. The MLP procedure supports multiple training algorithms. Use the `OPTIMIZER` option of the `MODEL` statement to select the training algorithm.

`OPTIMIZER=RPROP` selects a *batch* training algorithm, which processes the training data in a single block and can be expected to perform well on small datasets. Several *minibatch* training algorithms are also supported. Minibatch training algorithms process the training data in many small blocks and can be expected to perform well on large datasets. For the batch training algorithm, the network weights are updated when the whole batch has been processed. For the minibatch training algorithms the network weights are updated after each smaller block has been processed.

A single pass through the training data is known as an *epoch*.

The `MODEL` statement enables the specification of learning rate schedules with minibatch optimisers by using `LEARNINGRATES` in the `OPTIMIZER` option. Learning rate schedules typically reduce learning rates over time to achieve a good trade-off between speed of convergence and numerical stability.

The MLP procedure initialises the weights of a newly created network to random values as determined by the `INITWEIGHTS` option of the `MODEL` statement. When experimenting with different training algorithms and regularisation schemes, it is often useful to start from the same initial weight values. This can be achieved by specifying `SEED` in the `INITWEIGHTS` option of the `MODEL` statement.

Regularisation and performance assessment

To limit overfitting, a validation dataset can be specified using the `VALIDATION` option of `PROC MLP`. In this case, the validation dataset is used to assess the performance of the network during training. The frequency of assessment can be specified using the `VALIDATIONINTERVAL` option of the `MODEL` statement. The result of training is the network with the best validation set performance.

You can also control overfitting with the `REGULARIZE` option on the `MODEL` statement:

- You can use `REGULARIZE=LNNORM` to specify L1-norm and L2-norm regularisation. Typically, you use L1-norm regularisation for model selection, including the identification of irrelevant or redundant inputs, and L2-norm regularisation to control overfitting.
- You can use `REGULARIZE=DROPOUT` to specify dropout regularisation. Typically, you use dropout regularisation to control overfitting.

If the `VALIDATION` option is not specified, the result of training will be the network with the best regularised training error.

A test dataset can be specified using the `PROC MLP TEST` option. In this case, the test dataset is used to provide an unbiased estimate of generalisation performance when training terminates.

By default, for non-class response variables, the MLP procedure uses the squared error function to measure the errors in the network's predicted responses. For class response variables, the default error function is the cross entropy function. Alternative error functions can be specified using the `ERROR` option of the `MODEL` statement but the procedure will only run if the range of the network's output activation functions matches, or lies entirely within, the domain of the specified error function.

Terminating training

A variety of termination options can be specified in the `MODEL` statement, for example:

- the `MAXTRAININGTIME` option terminates training after a specific number of seconds
- the `MAXTRAININGEPOCH` option terminates training after the specified number of epochs
- the `TERMINATEONCOMPLETESEPARATION` option terminates training if the network succeeds in completely separating the different levels of the response variable in the training dataset
- the `MAXUNIMPROVEDVALIDATIONMINIBATCHES` option terminates training if the error on the validation set has not decreased for the specified number of minibatches

There are no default values for any of the termination options. If no termination options are specified, training continues indefinitely.

For further information about network training options, see `MODEL` [↗](#) (page 3269).

Loading and saving a trained network

Once you have trained a network there are several ways you can save and reuse it.

The `OUTEST` option of `PROC MLP` is used to specify a dataset to store the best network that was found during training. If the `VALIDATION` option is used, the best network will be the one with the lowest validation set error, otherwise it will be the one with the lowest regularized training set error.

A network can be loaded using the `INEST` option of `PROC MLP`, but the network must have the architecture specified in the `MODEL` statement. It must have the correct number of hidden layers and the correct number of neurons in each layer, and all neurons must have the correct activation functions.

The `INEST` option of `PROC MLP` can also be used to load a network for additional training; if the `NOFIT` option is also specified no training occurs, the network weights are not modified and only predicted responses and non-training related statistics are calculated. The `NOFIT` option can therefore be used to allow a trained network to be loaded and applied to a new dataset to generate predicted responses.

The `PROC MLP CODE` statement enables you to generate code that implements the trained network. This code can be used in a data step to score another dataset.

Results and generated output

PROC GMM outputs can be written to the log and to ODS output.

During training, the MLP procedure regularly records the training progress in the server log. Information includes the training error, the regularisation error (if the `REGULARIZER` option of the `MODEL` statement is specified) and the validation error (if a validation dataset is specified using the `VALIDATION` option of `PROC MLP`). The `TRAININGHISTORYUPDATEINTERVAL` option of the `MODEL` statement specifies how frequently the log messages are written. By default, a log message is written after every 1000 training epochs.

A summary of the training history can also be written to ODS output. This is configurable. As before, the `TRAININGHISTORYUPDATEINTERVAL` controls how often the training history is updated, and hence the training history entries that are available to be output. The `MAXTRAININGHISTORYSIZE` option of the `MODEL` statement specifies how many training history entries are reported in ODS output. The `COMPACTHISTORY` option of `PROC MLP` controls whether the output is an overall summary or a rolling history of the most recent training.

For more details see `PROC MLP` [↗](#) (page 3262) and `MODEL` [↗](#) (page 3269)

If the `OUTPUT` statement is specified, at the end of training, the MLP procedure generates an output dataset which contains the results of applying the best network found during training to the input dataset, the validation dataset (if present) and the test dataset (if present). For more information see `OUTPUT` [↗](#) (page 3295).

Using the MLP procedure

This example shows how to use `PROC MLP` to train a network from known data, then use it to predict results from new data.

This example describes how the MLP procedure can be applied to the publicly-available Iris dataset (Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), pp. 179–188.)

The Iris dataset consists of measurements of the widths and lengths of the petals and sepals of three species of Iris. `PROC MLP` is used to predict the species of iris from these measurements.

```
PROC MLP DATA=iris TEST=test PRINTWEIGHTS PRINTTHRESHOLDS THRESHOLDSTEPS=10
  COMPACTHISTORY;
  CLASS species;
  MODEL species=sepal_length sepal_width petal_length petal_width /
    HIDDEN=(2 LOGISTIC)
    OPTIMIZER=RPROP
    MINLEARNINGRATE=1e-6
    MAXTRAININGEPOCH=10000
    MAXTRAININGHISTORYSIZE=25
    TRAININGHISTORYUPDATEINTERVAL=1
    REGULARIZER=LNNORM(STRENGTH=0.075 POWER=2.0)
    INITWEIGHTS=(SEED=1494327418);
RUN;
```

In this example, the dataset is partitioned into a training dataset, `iris`, containing 120 observations and a test dataset, `test`, containing 30 observations.

The `PROC MLP` statement includes options to control the information that is output to ODS output:

- `PRINTWEIGHTS` specifies that output includes the weights between the network layers
- `PRINTTHRESHOLDS` specifies that the output includes the threshold tables.
- `THRESHOLDSTEPS=10` specifies that the threshold tables split the network output values into ten threshold values.
- `COMPACTHISTORY` specifies that the training history is compacted, so a summary of the complete history of network training is available. Otherwise, a rolling window of detailed training history is saved for output, but the later entries overwrite the earlier ones.

The `CLASS` statement defines the response variable `species` as a categorical variable.

The `MODEL` statement defines `sepal_length`, `sepal_width`, `petal_length` and `petal_width` as effect variables and `species` as the response variable. The `MODEL` statement includes the following options:

- The `HIDDEN` option is mandatory, and, in this example, specifies a single hidden layer which has two neurons that use the `LOGISTIC` activation function.
- The `OPTIMIZER` option specifies that the `RPROP` optimiser is used to train the network.
- The `REGULARIZER` option specifies that the `LNNORM` regulariser is used to prevent overfitting. The optimal value for `STRENGTH` was derived by previously partitioning the Iris dataset into a training dataset and a validation dataset, and repeatedly training the network with a range of different values for `STRENGTH`. The value selected to train the final network (0.75) is the value that minimised the validation set error reported in the Results table.
- The `MAXTRAININGHISTORYSIZE` and `TRAININGHISTORYUPDATEINTERVAL` options specify that up to 25 training history entries are reported in the ODS output, and that a training history entry is written for every epoch. The training history entry for each epoch is written to the server log, but, because `PROC MLP` includes the `COMPACTHISTORY` option, only a summary of the training history is written to ODS output.
- The `MINLEARNINGRATE` and `MAXTRAININGEPOCH` options specify that network training stops either when the learning rate drops below `1E-6` or when 10000 epochs have been completed. There are no default values for any of the termination options, so if no termination options are specified, training continues indefinitely.
- The `INITWEIGHTS` option specifies a seed for the random number generator that is used to initialise the network weights. This ensures that if the procedure is run again with the same input datasets the same results are obtained. The other `INITWEIGHTS` options have the default values (no bias offset, weights sampled from a normal distribution with mean 0.0 and variance 1.0, and Xavier scaling).

Configuration

The Configuration table contains the configuration that was specified for this execution of `PROC MLP`.

Configuration		
Setting		Value

Stopping criteria		
Maximum epoch	10000	
Minimum learning rate	1E-6	
Training parameters		
Error function	Cross entropy	
Regularizers		
Ln-norm1	0.075, 2	
Optimization algorithm	RProp	
eta+	1.2	
eta-	0.5	
maxdelta	0.1	
initialdelta	0.1	
Automate mindelta	Yes	
Weight initialization method	Uniform	
Weight initialization seed	1494327418	
Weight initialization min	-0.1	
Weight initialization max	0.1	
Preprocessor		
Mean	0	
Variance	1	

Network Architecture

The Network Architecture table contains information about the layers in the network, the number of neurons in each layer, and the type of activation function for each layer. In this example, there is just one hidden layer between the input and output layers. The input layer always uses a linear activation function. The hidden layer uses the `LOGISTIC` activation function as specified, and the output layer uses the `SOFTMAX` activation function, which is the default activation function type for categorical response variables.

Network Architecture		
Layer	Nodes	Type
Input Layer	4	Linear
Hidden layer	2	Logistic
Output Layer	3	Softmax

Training Thresholds

The Training Thresholds table shows the network response levels, and for each, the range of the values of the corresponding output neuron. For each range, the table shows the distribution of positive and negative samples across those values for the observations in the training dataset. The information in this table can be useful when deciding a threshold probability value to apply to the network output to ensure accurate categorisation.

In this example, for the `Iris-setosa` response level, the network has output a probability of between 0.0 and 0.1 for 80 negative observations (observations that are not in the `Iris-setosa` category) and a probability of between 0.9 and 1.0 for 40 positive observations (observations that are in the `Iris-setosa` category). This means that any threshold value between 0.1 and 0.9 can be applied to the network output for the `Iris-setosa` response level to separate the positive and negative observations.

The network outputs for the other two response levels (*Iris-versicolor* and *Iris-virginica*) cover a wider range of values. For the *Iris-versicolor* response level, the network has output a probability of between 0.0 and 0.1 for 74 negative observations (observations that are not in the *Iris-versicolor* category) and a probability of between 0.9 and 1.0 for 35 positive observations (observations that are in the *Iris-versicolor* category). But the other six observations that are not in the *Iris-versicolor* category and the other five observations that are in the *Iris-versicolor* category are scattered across the other output values. In particular, the network has output a probability between 0.3 and 0.4 for a negative observation, and for a positive observation, and also output a probability between 0.8 and 0.9 for a negative observation, and for a positive observation. In this case, there is no threshold value that can be applied to the network output for the *Iris-versicolor* response level to completely separate the positive and negative observations.

Training Thresholds				
Response level	Threshold range minimum	Threshold range maximum	Negative samples	Positive samples
<i>Iris-setosa</i>	0	0.1	80	0
	0.1	0.2	0	0
	0.2	0.3	0	0
	0.3	0.4	0	0
	0.4	0.5	0	0
	0.5	0.6	0	0
	0.6	0.7	0	0
	0.7	0.8	0	0
	0.8	0.9	0	0
<i>Iris-versicolor</i>	0.9	1	0	40
	0	0.1	74	0
	0.1	0.2	2	0
	0.2	0.3	2	0
	0.3	0.4	1	1
	0.4	0.5	0	1
	0.5	0.6	0	0
	0.6	0.7	0	1
	0.7	0.8	0	1
<i>Iris-virginica</i>	0.8	0.9	1	1
	0.9	1	0	35
	0	0.1	75	0
	0.1	0.2	1	1
	0.2	0.3	1	0
	0.3	0.4	1	0
	0.4	0.5	0	0
	0.5	0.6	1	0
	0.6	0.7	1	1
	0.7	0.8	0	2
	0.8	0.9	0	2
	0.9	1	0	34

Test Thresholds

The Test Thresholds table shows the network response levels, and for each, the range of the values of the corresponding output neuron, and the distribution of positive and negative samples across those values for the observations in the test dataset.

In this example, for the test dataset, in the `Iris-setosa` category section of the table, the network has classified the 20 observations that are not in the `Iris-setosa` category as having a probability of between 0.0 and 0.1 of belonging to this category. The network has classified the 10 observations that are in the `Iris-setosa` category as having a probability of between 0.9 to 1.0 of belonging to this category. Again, the outputs for the other two response levels cover a wider range of probabilities. But, in this case, for this small dataset, there are no overlaps, and it is possible to choose a threshold value to apply to the network output for each of the response levels to completely separate the positive and negative observations.

As with the training dataset, any threshold value between 0.1 and 0.9 could be applied to the network output for the `Iris-setosa` response level to separate the positive and negative observations. Based on this dataset, any threshold value between 0.5 and 0.9 could be used for the `Iris-versicolor` response level and any threshold value between 0.1 and 0.5 for the `Iris-virginica` response level.

Test Thresholds				
Response level	Threshold range minimum	Threshold range maximum	Negative samples	Positive samples
Iris-setosa	0	0.1	20	0
	0.1	0.2	0	0
	0.2	0.3	0	0
	0.3	0.4	0	0
	0.4	0.5	0	0
	0.5	0.6	0	0
	0.6	0.7	0	0
	0.7	0.8	0	0
	0.8	0.9	0	0
	0.9	1	0	10
Iris-versicolor	0	0.1	17	0
	0.1	0.2	0	0
	0.2	0.3	1	0
	0.3	0.4	1	0
	0.4	0.5	1	0
	0.5	0.6	0	0
	0.6	0.7	0	0
	0.7	0.8	0	0
	0.8	0.9	0	0
	0.9	1	0	10
Iris-virginica	0	0.1	20	0
	0.1	0.2	0	0
	0.2	0.3	0	0
	0.3	0.4	0	0
	0.4	0.5	0	0
	0.5	0.6	0	1
	0.6	0.7	0	2
	0.7	0.8	0	0
	0.8	0.9	0	0
	0.9	1	0	7

Training History

The Training History table shows how training, validation and regularisation errors change during training.

Training History				
Epoch	Training time(s)	Learning rate	Training data error	Regularization error
0	0	0.1	1.1659607314	0.0002823024
60	0	0.1	0.0750214228	0.0406435349
120	0	0.0089561962	0.0619656677	0.0466006967
184	0	0.0071868911	0.0619784137	0.0463739436
240	0	0.0009979837	0.0618877388	0.0464424851
304	0	0.002562157	0.0618779442	0.0464220304
368	0	0.000383699	0.0618617126	0.0464357259
416	0	0.0017706158	0.0617943858	0.0465021285
480	0	0.0004104514	0.061788151	0.046507944
544	0	0.0002045593	0.0617978459	0.0464981773
608	0	0.0000848182	0.061793415	0.0465025826
672	0	0.0000654333	0.0617906175	0.0465053712
736	0	4.7179414E-6	0.0617910668	0.0465049205
768	0	0.0000218068	0.0617906687	0.0465053182
832	0	9.4033976E-7	0.061790304	0.0465056828

Network Weights

The Network Weights table gives the names and values of all the weights in the network.

The names of the weights in the table are as follows:.

- Input layer to hidden layer weights:

```
wih-1-<hidden_neuron>-<input_neuron>
```

- hidden layer to hidden layer weights:

```
whh-<hidden_layer>-<postsynaptic_hidden_neuron>-<presynaptic_hidden_neuron>
```

This example only has one hidden layer so there are no hidden layer to hidden layer weights.

- Hidden layer to output layer weights:

```
wo-<output_neuron>-<hidden_neuron>
```

Network Weights

Weight	Value
wih-1-1-b	-4.915256
wih-1-1-1	-0.472136
wih-1-1-2	-0.698932
wih-1-1-3	3.1853894
wih-1-1-4	4.2123014
wih-1-2-b	2.7363908
wih-1-2-1	1.0886229
wih-1-2-2	-1.114099
wih-1-2-3	1.9501099
wih-1-2-4	1.7968779
wo-1-b	4.4582273
wo-1-1	-1.381585
wo-1-2	-5.59808
wo-2-b	-0.469427
wo-2-1	-4.544412
wo-2-2	4.5417894
wo-3-b	-2.007306
wo-3-1	5.9258882
wo-3-2	1.0562217

Results

The Results table displays the results of training.

Results

Result	Value
Last training set data error	0.0617903
Last partial separation	No
Last complete separation	No
Last Ln-norm1	0.0465057
Last total regularization error	0.0465057
Last total training set error	0.108296
Termination epoch	832
Termination time (s)	0
Test set error	0.0611409

Stopping Reason

The Stopping Reason table lists the reasons why training stopped. In this example, training stopped when the learning rate for the epoch was below the specified MINLEARNINGRATE of 1E-6.

Stopping Reasons

Reasons for stopping

Learning rate was below the termination threshold.

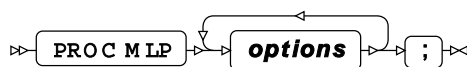
MLP procedure reference

Describes the syntax and options for `PROC MLP` and its contained statements.

<code>PROC MLP</code> ↗	3262
Creates a multilayer perceptron (MLP).	
<code>CLASS</code> ↗	3265
Specifies the class variables, and, optionally, how they are handled.	
<code>CODE</code> ↗	3267
Outputs the MLP as a file containing data step code, which can subsequently be used to score another dataset.	
<code>MODEL</code> ↗	3269
Specifies the response and effect variables and the options that control network training.	
<code>OUTPUT</code> ↗	3295
Creates an output dataset containing the input observations and the MLP outputs.	

PROC MLP

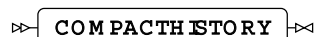
Creates a multilayer perceptron (MLP).



Options

The following *options* are available:

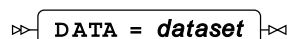
COMPACTHISTORY



Specifies that the training history is compacted when it reaches its maximum size, which allows the entire history of network training to be represented in a fixed amount of space.

If `COMPACTHISTORY` is not specified, then when the training history reaches its maximum size, the oldest entries are overwritten, and the history forms a moving window.

DATA



Specifies the dataset that is used for training.

If a training dataset is not specified, the most recently-created dataset is used.

INEST

» **INEST = *dataset*** «

Specifies a dataset that provides initial values for the network weights. The network in the dataset must have the same input encoding, output representation and structure as the network specified in the `MODEL` statement.

NOFIT

» **NOFIT** «

Specifies that `PROC MLP` should assess performance on the `DATA`, `VALIDATION` and `TEST` datasets without training the network. This is useful when loading a trained network using the `INEST` option.

NOPRINT

» **NOPRINT** «

Specifies that all ODS output is suppressed. This option takes precedence over all other print options.

OUTEST

» **OUTEST = *dataset*** «

Specifies the dataset used to store the input encoding, output representation, structure and weights of a trained network.

PRINTINPUTENCODING

» **PRINTINPUTENCODING** «

Specifies that the encoding of effect variables in terms of groups of network inputs is written to ODS output.

PRINTINPUTLENGTHS

» **PRINTINPUTLENGTHS** «

Specifies that the L2 norms of the weight to each input neuron is written to ODS output. This can be used with L1 regularisation to identify irrelevant inputs.

PRINTINPUTMAPPING

» **PRINTINPUTMAPPING** «

Specifies that the mapping of effect variable levels to individual network inputs is written to ODS output.

PRINTINPUTSCALING

⇒ **PRINTINPUTSCALING** ⇐

Specifies that the scaling of network input values are written to ODS output.

PRINTINPUTWEIGHTS

⇒ **PRINTINPUTWEIGHTS** ⇐

Specifies that the weights between the network inputs and the first layer of hidden nodes are written to ODS output.

PRINTTHRESHOLDS

⇒ **PRINTTHRESHOLDS** ⇐

Specifies that the thresholds table is written to ODS output.

The thresholds table shows the probabilities of the various values of the response variable and the numbers of observations predicted by the network to have those probabilities and response variable values.

The thresholds table can be used to determine a threshold value that balances the false positives and false negatives.

PRINTWEIGHTS

⇒ **PRINTWEIGHTS** ⇐

Specifies that the network weights are printed are written to ODS output.

TEST

⇒ **TEST = *dataset*** ⇐

Specifies a dataset to be used as a test dataset at the end of training.

THRESHOLDSTEPS

⇒ **THRESHOLDSTEPS** ⇒ **=** ⇒ ***threshold-steps*** ⇐

Specifies the number of discrete steps to show for the network output values in the thresholds table. The default value is 20.

The value must be an integer greater than 0 (zero).

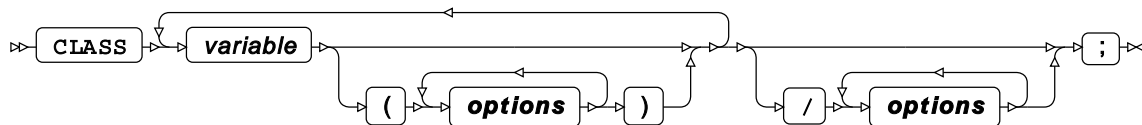
VALIDATION

➤ **VALIDATION = *dataset*** ➤

Specifies a validation dataset. If a validation dataset is specified, the result of training is the network that achieved the minimum error against this dataset.

CLASS

Specifies the class variables, and, optionally, how they are handled.



A class (or categorical) variable is a variable that can take one of a limited number of values.

The options, if present, define the way the class variables are handled in the model. Options can be specific to a single variable or can be global. Options in brackets after a variable name are specific to that variable. Options specified after the forward slash are global options and apply to all class variables. Variable-specific options override global options. Unless otherwise specified, all options can be variable-specific or global.

The response variable always uses GLM encoding. Effect variables use GLM encoding unless an alternative encoding is specified using the `PARAM` option. If an alternative encoding is specified for any effect variable, all effect variables must use non-GLM encodings, but all effect variables do not need to use the same non-GLM encoding.

If present, the `CLASS` statement or statements must be located before the `MODEL` statement.

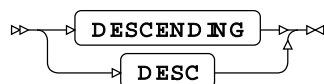
variable

A class variable in the dataset.

Options

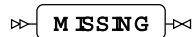
The following *options* are available. Unless otherwise stated, all options can be applied to a single variable, or globally to all variables.

DESCENDING



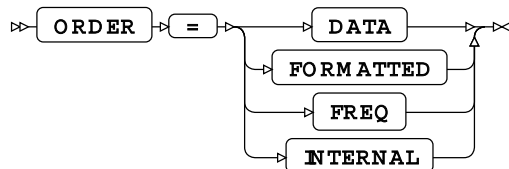
Specifies that the variable values are sorted in descending order. If not specified, the variable values are sorted in ascending order. The way in which the variable values are ordered is specified using the `ORDER` option.

MISSING



Specifies that a level is created for missing values and that observations containing missing values are retained. If not specified, observations with missing values are discarded.

ORDER



Specifies the ordering to use for variable values. Variable values are ordered in ascending order unless the `DESCENDING` option is also specified.

For numeric variables, the default ordering is `INTERNAL`. For string variables, the default ordering is `FORMATTED`.

DATA

The variable values are ordered in the order in which the values of the variable first occur in the data.

FORMATTED

The variable has a user-defined format applied, and the variable values are ordered using the variable format value.

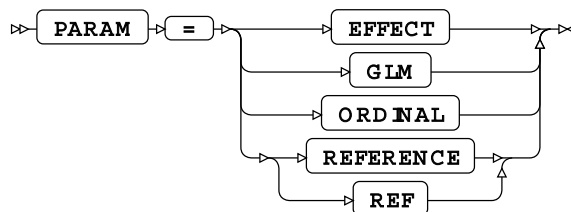
FREQ

If the `FREQ` ordering is specified, it is ignored, and the default ordering of `INTERNAL` (for numeric variables) or `FORMATTED` (for string variables) is used.

INTERNAL

The variable is unformatted, and the variable values are ordered by the raw value.

PARAM



The default encoding is `GLM`. If a non-GLM encoding is specified, then all effect (or predictor) variables must use non-GLM encoding. Response variables always use GLM encoding.

Response variables always use GLM encoding. The default encoding for effect (or predictor) variables is GLM encoding. If a non-GLM encoding is specified for an effect variable, then all effect variables must use non-GLM encoding.

EFFECT

Specifies that the variable is encoded using effect encoding.

GLM

Specifies that the variable is encoded using GLM encoding.

This value can only be specified as a global option and not for an individual variable.

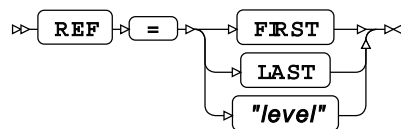
ORDINAL

Specifies that the variable is encoded using ordinal encoding.

REFERENCE

Specifies that the variable is encoded using reference encoding.

REF



Specifies the reference level to use for a class variable.

FIRST

The reference level is the first ordered level.

LAST

The reference level is the last ordered level. This is the default reference level.

"level"

Specifies a value to use as the reference level. The specified value must be a valid value for the class variable.

CODE

Outputs the MLP as a file containing data step code, which can subsequently be used to score another dataset.



When the code file has been produced, you can write a data step that contains a `SET` statement to specify the dataset to be scored, followed by a `%INCLUDE` statement to include the code file containing the MLP. The output dataset contains the scored data.

FILE

Specifies the name of the file to contain the code.

Options

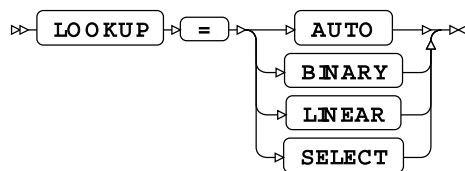
The following *options* are available:

LINESIZE

➤ `LINESIZE = length` ➤

Specifies the maximum line length in the output file.

LOOKUP



Specifies the method used to lookup and encode class variable levels.

AUTO

Specifies that `PROC MLP` selects the best method (`BINARY`, `LINEAR`, `SELECT`) to use for the data.

BINARY

Specifies that a binary search is used.

LINEAR

Specifies that the data is searched starting at first observation and progressing through the class levels in order until the value is located.

SELECT

Specifies that the `DATA` step `SELECT` functionality is used.

OUTPUTHIDDENACTIVITIES

➤ `OUTPUTHIDDENACTIVITIES` ➤

Specifies that the generated code outputs the activities of the hidden neurons.

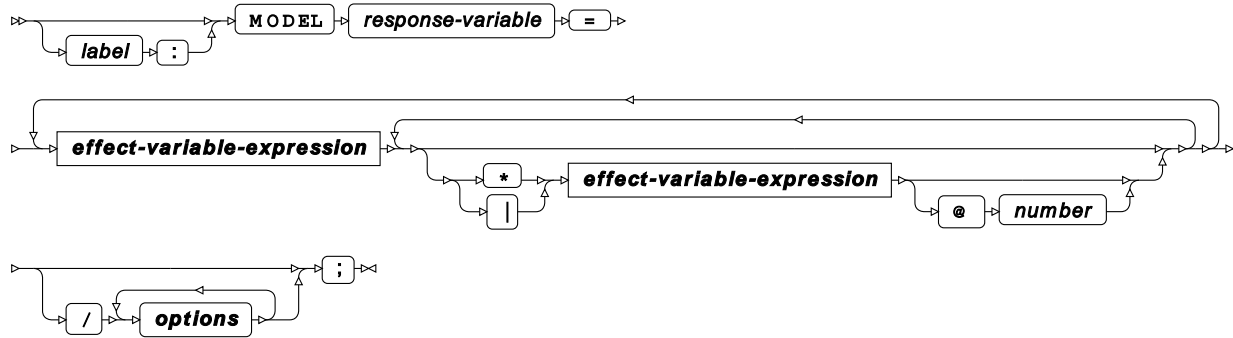
OUTPUTINPUTACTIVITIES

➤ `OUTPUTINPUTACTIVITIES` ➤

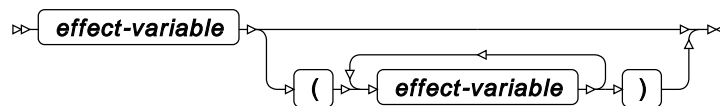
Specifies that the generated code outputs the activities of the input neurons.

MODEL

Specifies the response and effect variables and the options that control network training.



effect-variable-expression



response-variable

Specifies the response variable in the input dataset. The specified variable must also be present in the test dataset (if used) and the validation dataset (if used).

effect-variable

Specifies a variable in the input dataset to include in the model as an effect variable, and optionally, specifies how it is combined with other variables to derive additional effect variables to include in the model.

The specified variable must also be present in the test dataset (if used) and the validation dataset (if used).

The effect variables can be combined in the same way as variables in other regression procedures:

- * Include a new variable that is the product of the specified variables.
For example, `var1*var2` defines an effect variable that is the product of `var1` and `var2`.
- | Include each specified variable. Also include new variables from the products of each possible combination of two or more of the specified variables .

For example, `var1 | var2 | var3` defines the effect variables `var1`, `var2`, `var3`, `var1*var2`, `var1*var3`, `var2*var3` and `var1*var2*var3`.

- @ When combining multiple variables to make a new variable, include no more than the specified number of variables in each combination.

For example, `var1 | var2 | var3@2` defines the effect variables `var1`, `var2`, `var3`, `var1*var2`, `var1*var3` and `var2*var3`, but not `var1*var2*var3`.

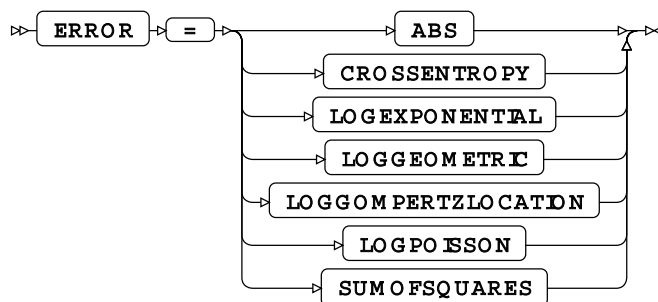
- () The variable outside the brackets is nested on all possible discrete values of the variable inside the brackets, and each nesting defines a new effect variable. The variable inside the brackets must be a discrete variable specified in a `CLASS` statement.

For example, if `var2` has values A, B and C, `var1 (var2)` defines the effect variables `var1 (A)`, `var1 (B)` and `var1 (C)`.

Options

The following *options* are available:

ERROR



Specifies the type of error function used to assess performance. The specified error function must be compatible with the output layer activation function and the response variable type.

The range of the output layer activation function specified using the `OUTPUT` option of the `MODEL` statement must not exceed the domain of the error function specified using the `ERROR` option of the `MODEL` statement. For example, `OUTPUT=LINEAR` cannot be used with `ERROR=CROSSENTROPY` because the `LINEAR` function can generate positive and negative numbers, but the `CROSSENTROPY` function can only process positive numbers.

For classification response variables, the default value is `CROSSENTROPY`. For non-classification response variables, the default value is `SUMOFSQUARES`.

ABS

Specifies that the absolute error function is used. This estimates the conditional median of the data.

CROSSENTROPY

Specifies that the cross entropy error function is used. This estimates the probability parameters of binomial or multinomial distributed data.

LOGEXPONENTIAL

Specifies that the log-exponential error function is used. This estimates the rate parameter of exponentially-distributed data.

LOGGEOMETRIC

Specifies that the log-geometric error function is used. This estimates the probability parameter of geometrically-distributed data.

LOGGOMPERTZLOCATION

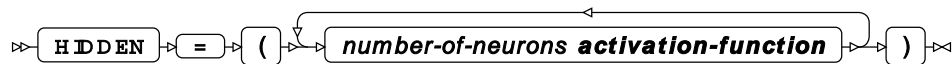
Specifies that the log-Gompertz error function is used. This estimates the location parameter of Gompertz-distributed data.

LOGPOISSON

Specifies that the log-Poisson error function is used. This estimates the rate parameter of Poisson-distributed data.

SUMOF SQUARES

Specifies that the sum of squares error function is used. This estimates the conditional mean of the data.

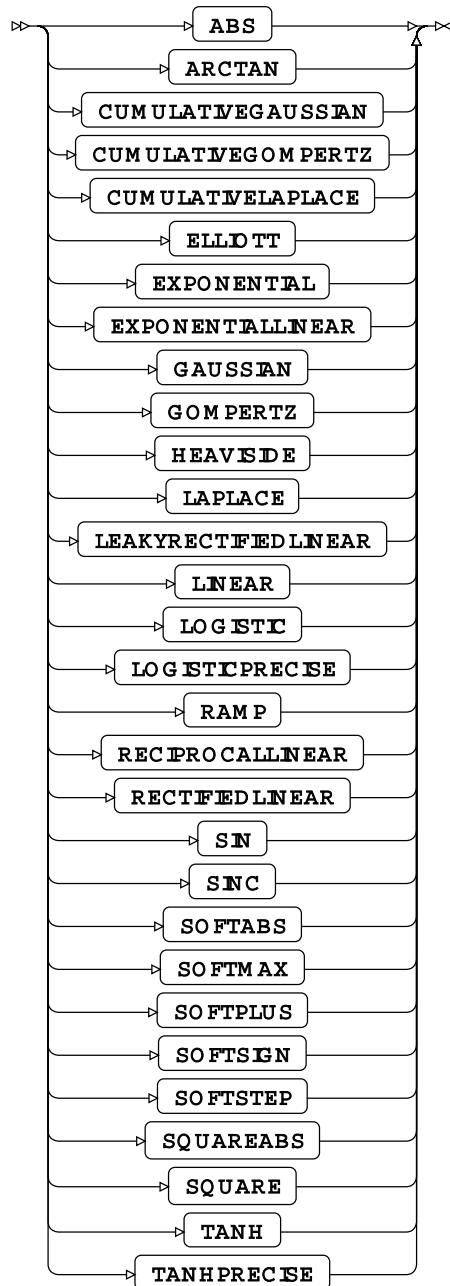
HIDDEN

Specifies the structure of the hidden layers. There is one *number-of-neurons* and *activation-value* pair for each hidden layer in the network. The first pair of values in the list specifies the hidden layer immediate below the input layer and the last pair of values in the list specifies the hidden layer immediately above the output layer.

The **HIDDEN** option must be specified exactly once in the **MODEL** statement.

number-of-neurons

Specifies the number of neurons in a hidden layer. Must be an integer, greater than or equal to 1.

activation-function

Specifies the activation function for a hidden layer.

For more information about the activation functions, see *Activation functions* [↗](#) (page 3293).

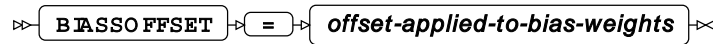
INITWEIGHTS

Specifies the method used to initialize network weights.

initweight-options

The following *initweight-options* are supported.

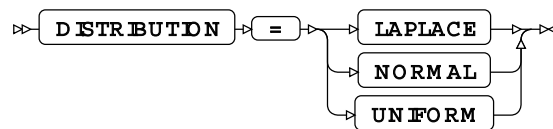
BIASSOFFSET



Specifies an additive offset for bias weights. This is typically set to a small positive value when RECTIFIEDLINEAR units are used to reduce the risk of creating dead neurons.

The default value is 0.0.

DISTRIBUTION



Specifies the type of distribution from which weights are sampled.

LAPLACE

$$p(x) = \frac{1}{\sqrt{2 \text{ VARIANCE}}} \exp \left\{ -\frac{|x - \text{MEAN}| \sqrt{2}}{\sqrt{\text{VARIANCE}}} \right\}$$

NORMAL

$$p(x) = \frac{1}{\sqrt{2\pi \text{ VARIANCE}}} \exp \left\{ -\frac{(x - \text{MEAN})^2}{2 \text{ VARIANCE}} \right\}$$

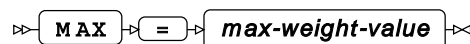
This is the default value.

UNIFORM

$$p(x) = \frac{1}{\text{MAX} - \text{MIN}} \text{ if } \text{MIN} \leq x < \text{MAX}$$

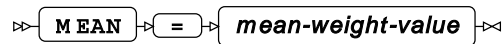
$$p(x) = 0 \text{ otherwise}$$

MAX



Specifies the maximum initial weight value for the UNIFORM distribution. The specified value must be greater than MIN.

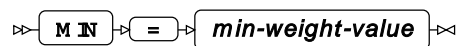
This option can only be specified with DISTRIBUTION=UNIFORM. The default value is 0.1.

MEAN

Specifies the mean for the `NORMAL` and `LAPLACE` distributions.

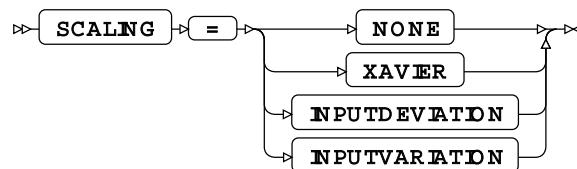
If `DISTRIBUTION` is not one of these values, this option is ignored.

The default value is 0.0.

MIN

Specifies the minimum initial weight value for the `UNIFORM` distribution. The specified value must be less than `MAX`.

This option can only be specified with `DISTRIBUTION=UNIFORM`. The default value is -0.1.

SCALING

Specifies how the magnitudes of the initial weights are scaled to compensate for different numbers of neurons in each layer.

INPUTDEVIATION

Scales the initial weights to control the standard deviation of the activation of neurons.

INPUTVARIATION

Scales the initial weights to control the variance of the activation of neurons.

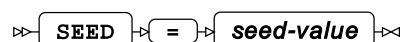
NONE

Weight values are not scaled.

XAVIER

Performs Xavier scaling that is appropriate for `TANH` activation functions.

This is the default value.

SEED

Specifies a seed value for the random number generator that is used to initialize the weights. This value must be an integer greater than or equal to 0 (zero).

The default value is a random value derived from the system clock.

VARIANCE

» **VARIANCE** = *variance-of-weight-values* «

Specifies the variance of the `NORMAL` and `LAPLACE` distributions. This value must be greater than or equal to 0 (zero).

If `SCALING=XAVIER` is specified, the default value is 1.0. Otherwise the default value is 0.1.

MAXFAILEDLIKELIHOODEPOCHS

» **MAXFAILEDLIKELHOODEPOCHS** = *max-failed-likelihood-steps* «

Specifies that training terminates if the likelihood cannot be computed for the specified number of successive epochs. Failures to compute likelihood can usually be eliminated by using smaller learning rates or L2-norm regularisation. This value must be an integer greater than 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXINCREASINGVALIDATIONEPOCHS

» **MAXINCREASNGVALDATIONEPOCHS** = *max-increasing-validation-epochs* «

Specifies that training terminates if the validation error increases for the specified number of successive epochs. This value must be an integer greater than 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXINCREASINGVALIDATIONMINIBATCHES

» **MAXINCREASNGVALDATIONMINIBATCHES** = *max-increasing-validation-minibatches* «

Specifies that training terminates if the validation error increases for the specified number of successive minibatches. This value must be an integer greater than 0 (zero).

If a batch training algorithm is specified, this option is equivalent to `MAXINCREASINGVALIDATIONEPOCHS`.

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXTRAININGEPOCH

» **MAXTRAININGEPOCH** = *max-epoch* «

Specifies that training terminates when the specified number of epochs have been completed. This value must be an integer greater than or equal to 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXTRAININGHISTORYSIZE

➤ `MAXTRAININGHISTORYSIZE` ➤ = ➤ `max-training-history-size` ➤

Specifies the maximum number of entries in the training history. This value must be an integer greater than 0 (zero).

MAXTRAININGMINIBATCH

➤ `MAXTRAININGMINIBATCH` ➤ = ➤ `max-minibatch` ➤

Specifies that training terminates when the specified number of minibatches have been completed. This value must be an integer greater than or equal to 0 (zero).

If a batch training algorithm is specified, this option is equivalent to `MAXTRAININGEPOCH`.

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXTRAININGTIME

➤ `MAXTRAININGTIME` ➤ = ➤ `max-training-time` ➤

Specifies that training terminates after the specified time in seconds. Training stops at the end of the first epoch or minibatch at or after this time. This value must be an integer greater than 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXUNIMPROVEDVALIDATIONEPOCHS

➤ `MAXUNIMPROVEDVALIDATIONEPOCHS` ➤ = ➤ `max-unimproved-validation-epochs` ➤

Specifies that training terminates if the validation error does not decrease for the specified number of epochs. This value must be an integer greater than 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MAXUNIMPROVEDVALIDATIONMINIBATCHES

➤ `MAXUNIMPROVEDVALIDATIONMINIBATCHES` ➤ = ➤ `max-unimproved-validation-minibatches` ➤

Specifies that training terminates if the validation error does not decrease for the specified number of minibatches. This value must be an integer greater than 0 (zero).

If a batch training algorithm is specified, this option is equivalent to

MAXUNIMPROVEDVALIDATIONEPOCHS.

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MINABSOLUTEWEIGHTCHANGE

⇒ **MINABSOLUTEWEIGHTCHANGE** ⇒ = ⇒ *min-absolute-weight-change* ⇒

Specifies that training terminates if no weight has changed by more than the specified amount in one minibatch. This value must be greater than or equal to 0 (zero). This option is ignored if the OPTIMIZER=RPROP is specified.

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MINLEARNINGRATE

⇒ **MINLEARNINGRATE** ⇒ = ⇒ *min-learning-rate* ⇒

Specifies that training terminates if the learning rate or step size falls below the specified value. This value must be greater than or equal to 0 (zero).

There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MINRELATIVEWEIGHTCHANGE

⇒ **MINRELATIVEWEIGHTCHANGE** ⇒ = ⇒ *min-relative-weight-change* ⇒

Specifies that training terminates if no weight changes by more than the specified amount relative to its previous value plus MINRELATIVEWEIGHTCHANGEEPSILON in one minibatch. This value must be greater than or equal to 0 (zero). This option is ignored if the OPTIMIZER=RPROP is specified.

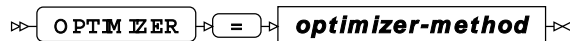
There is no default value for this option. If none of the termination criteria options are specified, training continues indefinitely.

MINRELATIVEWEIGHTCHANGEEPSILON

⇒ **MINRELATIVEWEIGHTCHANGEEPSILON** ⇒ = ⇒ *min-relative-weight-change-epsilon* ⇒

Specifies a value that is added to the previous absolute value of a weight when applying the MINRELATIVEWEIGHTCHANGE criterion. If no value is specified, the default value is machine epsilon. This value must be greater than 0 (zero).

OPTIMIZER



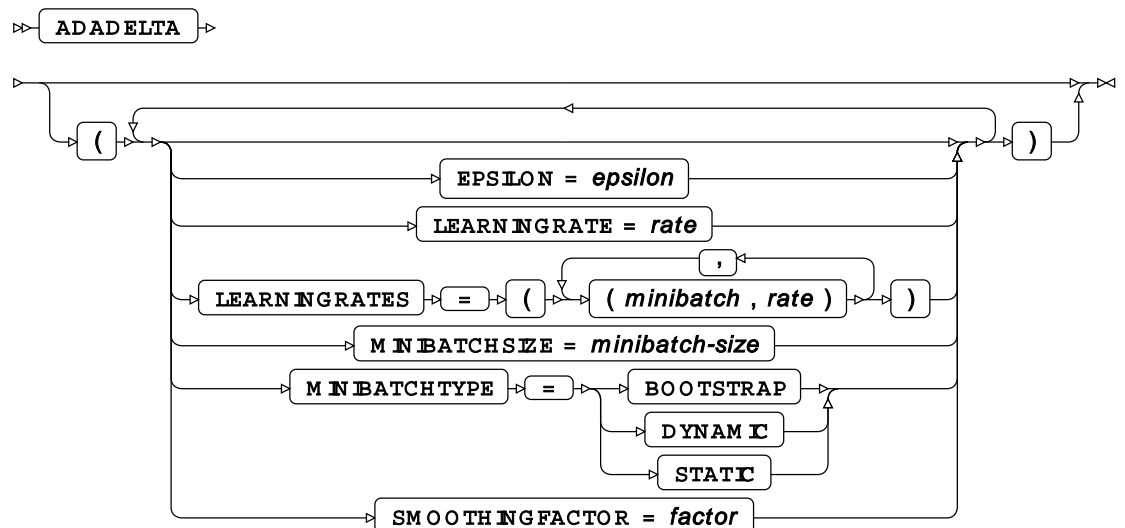
Specifies the optimization algorithm to be used to train the network.

The default *optimizer-method* is `ADADELTA`.

optimizer-method

The following *optimizer-method* options are supported:

- `ADADELTA` [↗](#) (page 3278). Specifies an adaptive step size minibatch learning algorithm that reduces sensitivity to the choice of learning rate.
- `ADAM` [↗](#) (page 3280). Attempts to maximize speed of convergence by using estimates of lower-order moments.
- `NADAM` [↗](#) (page 3282). A version of the `ADAM` optimiser method that incorporates Nesterov momentum.
- `RMSPROP` [↗](#) (page 3283). An adaptive step size minibatch learning algorithm.
- `RPROP` [↗](#) (page 3285). A variant of the adaptive step size full batch learning algorithm *iRProp+*.
- `SGD` [↗](#) (page 3286). A fixed step size minibatch learning algorithm that supports both classical and Nesterov momentum.
- `SMORMS3` [↗](#) (page 3287). An adaptive step size minibatch learning algorithm that automatically adjusts the amount of smoothing.

ADADELTA

Specifies an adaptive step size minibatch learning algorithm that reduces sensitivity to the choice of learning rate. Minibatch learning algorithms are best suited to training networks on large or highly redundant datasets.

EPSILON

Specifies a value for the epsilon parameter. This value must be greater than or equal to 0 (zero).

The default value for `OPTIMIZER=ADADELTA` is 1.0E-6.

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

Specifies that observations are selected randomly with replacement.

DYNAMIC

Specifies that observations are selected randomly without replacement.

This is the default value.

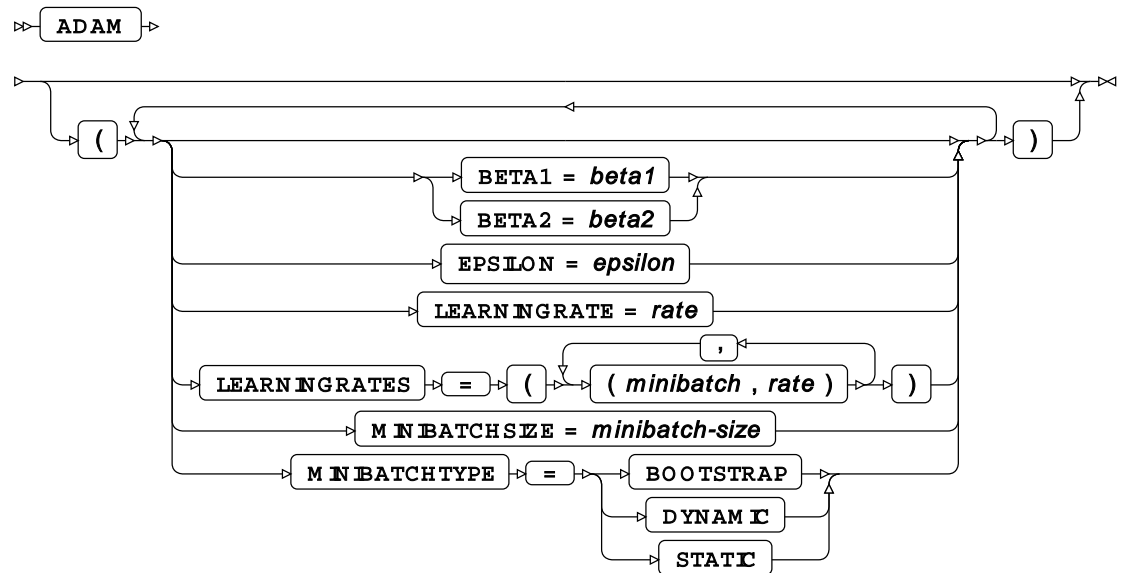
STATIC

Specifies that observations are selected in the order they appear in the input dataset.

SMOOTHINGFACTOR

Specifies a value for the smoothing factor. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value for OPTIMIZER=ADADELTA is 0.95.

ADAM

Specifies an adaptive step size minibatch learning algorithm that attempts to maximize speed of convergence by using estimates of lower-order moments. Minibatch learning algorithms are best suited to training networks on large or highly redundant datasets.

BETA1

Specifies a value for the beta1 parameter. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value is 0.9.

BETA2

Specifies a value for the beta2 parameter. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value is 0.999.

EPSILON

Specifies a value for the epsilon parameter. This value must be greater than or equal to 0 (zero)

The default value for OPTIMIZER=ADAM is 1.0E-8.

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

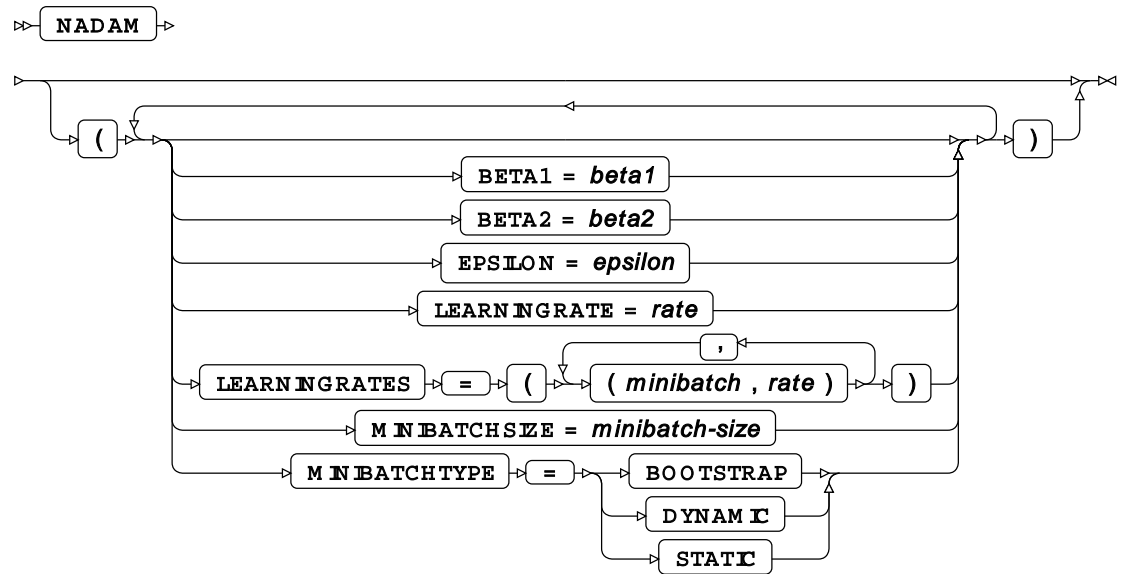
Specifies that observations are selected randomly with replacement.

DYNAMIC

Specifies that observations are selected randomly without replacement.

STATIC

Specifies that observations are selected in the order they appear in the input dataset.

NADAM

Specifies a version of the `ADAM` optimiser method that incorporates Nesterov momentum. Minibatch learning algorithms are best suited to training networks on large or highly redundant datasets.

BETA1

Specifies a value for the beta1 parameter. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value is 0.9.

BETA2

Specifies a value for the beta2 parameter. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value is 0.999.

EPSILON

Specifies a value for the epsilon parameter. This value must be greater than or equal to 0 (zero)

The default value for `OPTIMIZER=NADAM` is 1.0E-8.

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

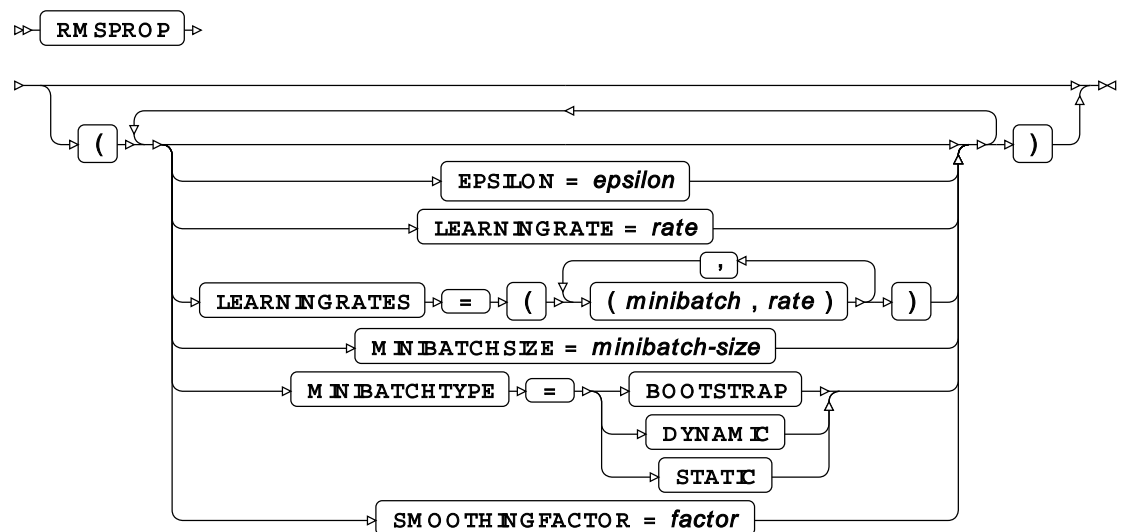
Specifies that observations are selected randomly with replacement.

DYNAMIC

Specifies that observations are selected randomly without replacement.

STATIC

Specifies that observations are selected in the order they appear in the input dataset.

RMSPROP

Specifies an adaptive step size minibatch learning algorithm.

EPSILON

Specifies a value for the epsilon parameter. This value must be greater than or equal to 0 (zero)

The default value for `OPTIMIZER=RMSPROP` is 1.0E-8.

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

Specifies that observations are selected randomly with replacement.

DYNAMIC

Specifies that observations are selected randomly without replacement.

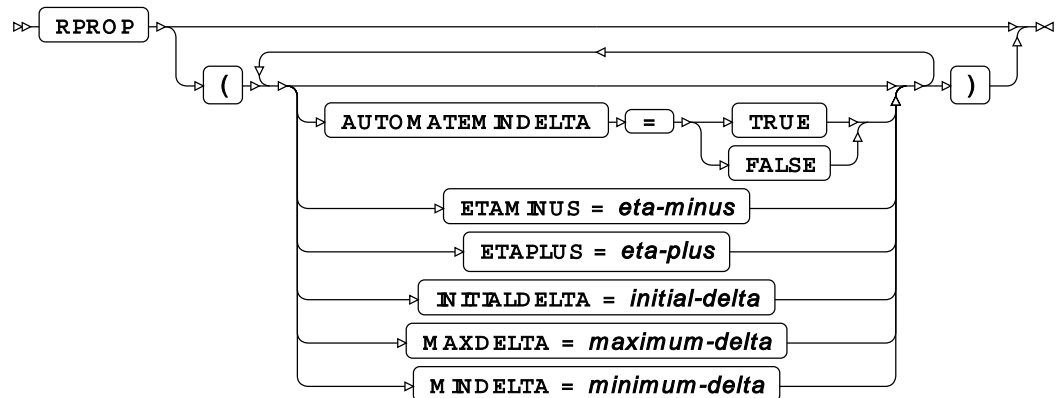
STATIC

Specifies that observations are selected in the order they appear in the input dataset.

SMOOTHINGFACTOR

Specifies a value for the smoothing factor. This value must be greater than or equal to 0 (zero) and less than or equal to 1.

The default value for OPTIMIZER=RMSPROP is 0.9.

RPROP

Specifies a variant of the adaptive step size full batch learning algorithm *iRProp+*. Full batch learning algorithms are best suited to training networks on datasets that are small or have very little redundancy.

AUTOMATEMINDELTA

Specifies the minimum step size. If no value is specified, the default is **TRUE**.

FALSE

Specifies **MINDELTA** as the minimum step size for all weights.

TRUE

Specifies that the minimum step size should vary with parameter size.

ETAMINUS

Specifies the factor by which the step size is reduced. This value must be greater than or equal to 0 (zero). If no value is specified, the default of 0.5 is used.

ETAPLUS

Specifies the factor by which the step size is increased. This value must be greater than or equal to 0 (zero). If no value is specified, the default of 1.2 is used.

INITIALDELTA

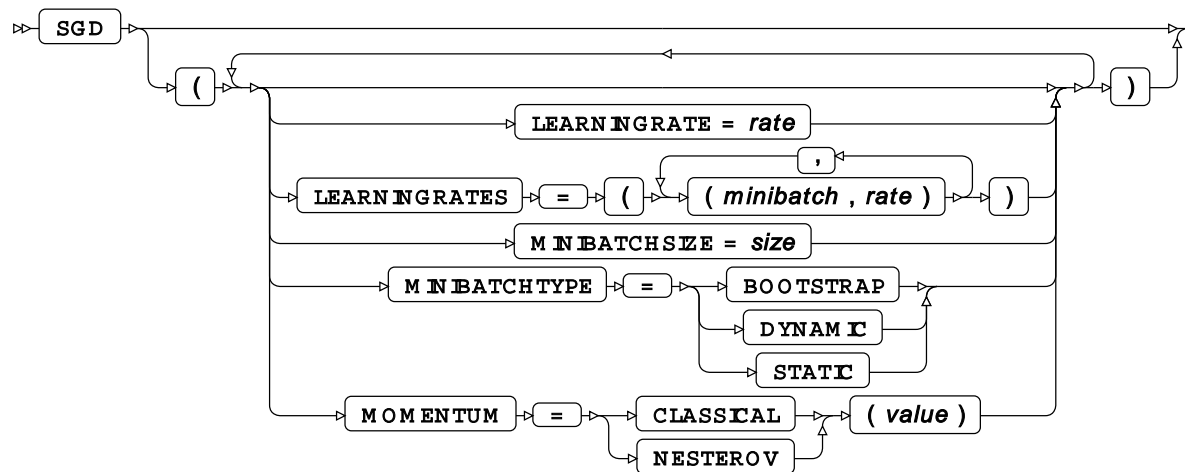
Specifies the initial step size. This value must be greater than or equal to 0 (zero). If no value is specified, the default of 0.1 is used.

MAXDELTA

Specifies the maximum step size. This value must be greater than or equal to 0 (zero). If no value is specified, the default of 0.1 is used.

MINDELTA

Specifies the minimum step size. This value must be greater than or equal to 0 (zero). If not specified, the value of `DBL_MIN` (the smallest positive double precision floating point value) is used.

SGD

Specifies a fixed step size minibatch learning algorithm that supports both classical and Nesterov momentum. Minibatch learning algorithms are best suited to training networks on large or highly redundant datasets.

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

Specifies that observations are selected randomly with replacement.

DYNAMIC

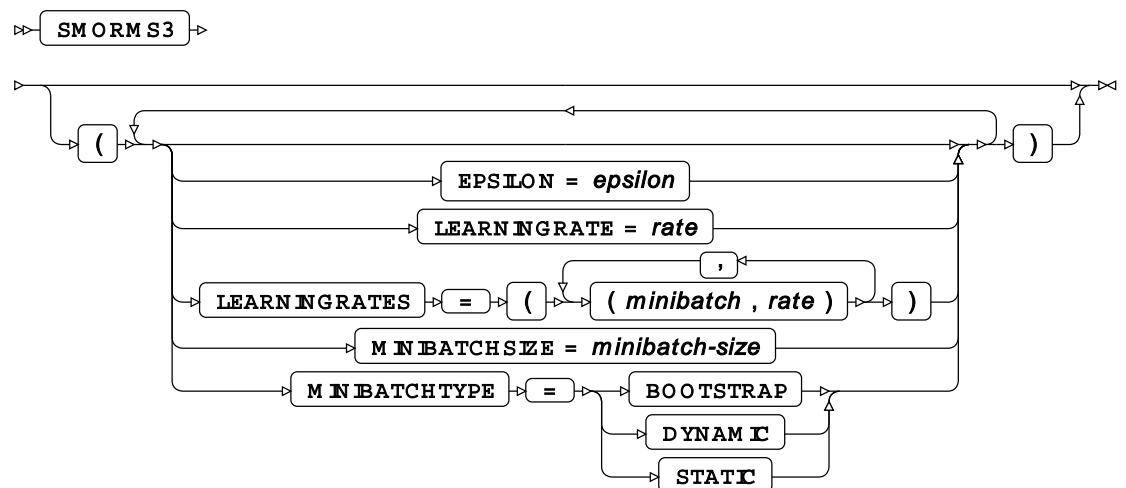
Specifies that observations are selected randomly without replacement.

STATIC

Specifies that observations are selected in the order they appear in the input dataset.

MOMENTUM

Specifies the type of momentum and the value of the momentum parameter.

SMORMS3

Specifies an adaptive step size minibatch learning algorithm that automatically adjusts the amount of smoothing.

EPSILON

Specifies a value for the epsilon parameter. This value must be greater than or equal to 0 (zero)

LEARNINGRATE

Specifies a fixed learning rate.

This option cannot be specified with `LEARNINGRATES`. This value must be greater than or equal to 0 (zero).

The default value is 0.001.

LEARNINGRATES

Specifies a learning rate schedule as a series of number pairs. Each pair contains a minibatch number and a learning rate. The pairs must be specified in order of increasing minibatch number. The learning rates are linearly interpolated between adjacent number pairs.

This option cannot be specified with `LEARNINGRATE`. The value of each number in the pair must be greater than or equal to 0 (zero).

MINIBATCHSIZE

Specifies the number of observations in each minibatch. This value must be an integer greater than 0 (zero) and less than or equal to `INT_MAX`.

The default value is 128.

MINIBATCHTYPE

Specifies how observations are selected for minibatches.

The default selection method is `DYNAMIC`.

BOOTSTRAP

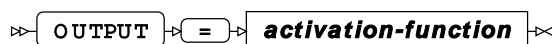
Specifies that observations are selected randomly with replacement.

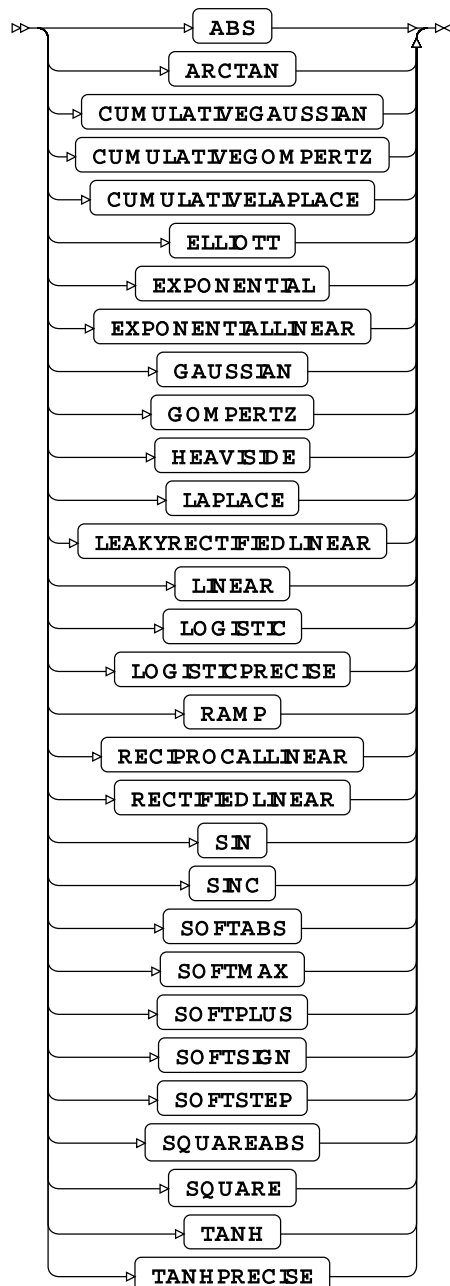
DYNAMIC

Specifies that observations are selected randomly without replacement.

STATIC

Specifies that observations are selected in the order they appear in the input dataset.

OUTPUT

activation-function

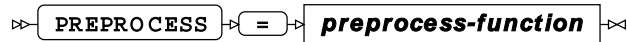
Specifies the activation function for the output layer.

The range of the output layer activation function specified using the `OUTPUT` option of the `MODEL` statement must not exceed the domain of the error function specified using the `ERROR` option of the `MODEL` statement. For example, `OUTPUT=LINEAR` cannot be used with `ERROR=CROSSENTROPY` because the `LINEAR` function can generate positive and negative numbers, but the `CROSSENTROPY` function can only process positive numbers.

For classification response variables, the default value is `SOFTMAX`. For non-classification response variables, the default value is `LINEAR`.

For more information about the activation functions, see [Activation functions](#) (page 3293).

PREPROCESS



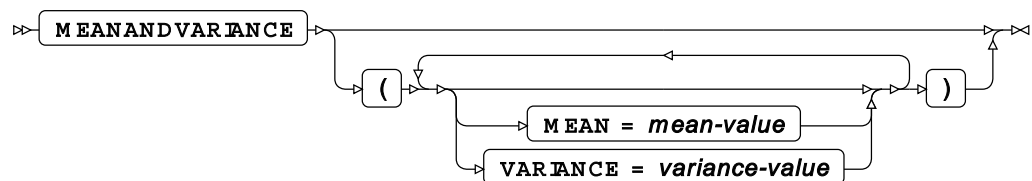
Specifies the preprocessing function to be applied to non-categorical effect variables.

The default function is MEANANDVARIANCE.

preprocess-function

The following *preprocess-function* options are supported.

MEANANDVARIANCE



Specifies that the values of each input variable are scaled to have the specified MEAN and VARIANCE.

MEAN

The mean of the transformed variable values.

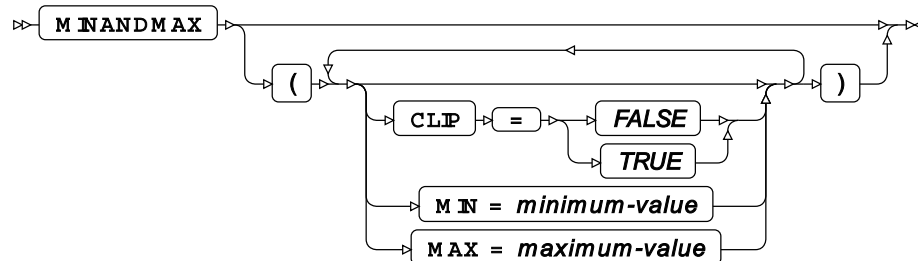
The default value is 0.0.

VARIANCE

The variance of the transformed variable values.

The default value is 1.0.

MINANDMAX



Specifies that the values of each effect variable are linearly transformed to lie between the specified minimum and maximum values. The minimum value of each variable in the training dataset is scaled to MIN, the maximum value is scaled to MAX and all other values are linearly interpolated between these two values.

CLIP

Specifies how new values of each input variable are transformed if they lie below the minimum value found for the variable in the original training dataset, or above the maximum value found for the variable in the original training dataset. **CLIP=TRUE** specifies that all values below the minimum value are scaled to **MIN** and all values above the maximum value are scaled to **MAX**. **CLIP=FALSE** specifies that values outside the range found in the training dataset are linearly extrapolated using the same scaling as originally used.

The default value is **FALSE**.

MAX

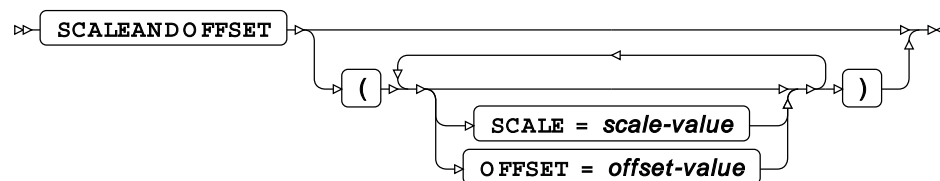
The maximum value for the transformed variable values.

The default value is 1.0.

MIN

The minimum value for the transformed variable values.

The default value is -1.0.

SCALEANDOFFSET

Specifies that the values of each input variable are transformed by subtracting the offset value from the variable value and then multiplying it by the scale value.

SCALE

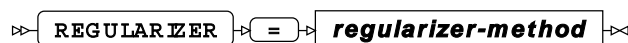
The scale value to use for the transformation. This value must be greater than or equal to **-DBL_MAX** and less than or equal to **DBL_MAX**. **SCALE** cannot be equal to 0 (zero).

The default value is 1.0.

OFFSET

The offset value to use for the transformation. This value must be greater than or equal to **-DBL_MAX** and less than or equal to **DBL_MAX**.

The default value is 0.0.

REGULARIZER

Specifies the type of regularisation to be used during training. You can use multiple **REGULARIZER** statements, so that more than one type of regularisation can be applied simultaneously.

If this option is not specified, no regulariser is used.

DROPOUT



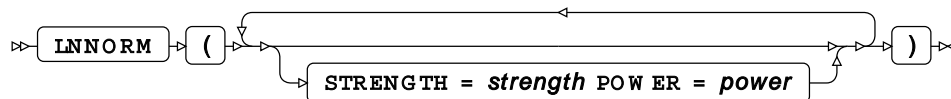
Specifies that dropout regularisation should be used on all hidden layers and that neurons will be dropped out during training with the specified probability. A seed for the dropout process can be specified to ensure repeatability. Only one dropout regulariser can be specified and dropout regularisation may only be used with minibatch optimizers.

PROBABILITY

Specifies the probability of neurons being dropped out during training. This value must be greater than 0 (zero) and less than 1.

The default value is 0.5.

LNNORM



Specifies that a norm-based regulariser be applied to all non-bias weights. The `STRENGTH` option specifies the strength of the regulariser and the `POWER` option specifies its power. A strong L2-norm regulariser, for example, might have `STRENGTH=1.0` and `POWER=2.0`, while a weak L1-norm regulariser might have `STRENGTH=1.0E-6` and `POWER=1.0`.

STRENGTH

Specifies the strength of the regulariser. This value must be greater than 0 (zero).

The default value is 0.0, which means there is no regularisation effect.

POWER

Specifies the power of the regulariser. This value must be greater than 0 (zero).

The default value is 2.0.

TERMINATEONCOMLETESEPARATION



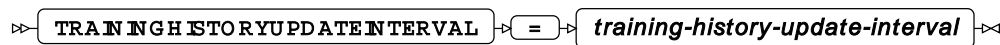
Specifies whether training terminates if the network is able to completely separate the different levels of the response variable in the training dataset.

By default, training is not terminated if the network reaches the state where the training data is completely separated.

TERMINATEONPARTIALSEPARATION

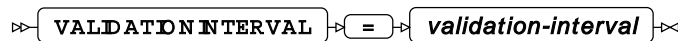
Specifies whether training terminates if the network is able to partially separate the different levels of the response variable in the training dataset (that is, the observations are either completely separated by a threshold value, or are on the threshold value).

By default, training is not terminated if the network reaches the state where the training data is partially separated.

TRAININGHISTORYUPDATEINTERVAL

Specifies the interval between epochs in the training history record.

The default value is 1000.

VALIDATIONINTERVAL

Specifies the interval between validation set assessments. If `OPTIMIZER = RPROP` is selected, the interval is interpreted as a number of epochs. For all other optimisers, the interval is interpreted as a number of minibatches. This value must be an integer greater than 0 (zero).

The default validation interval value is 1.

Activation functions

An activation function specifies how the activity y of a neuron is calculated from its post-synaptic potential x .

The following activation functions are available:

ABS $y = |x|$

ARCTAN $y = \arctan(x)$

CUMULATIVEGAUSSIAN $y = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right]$

CUMULATIVEGOMPertz $y = 1 - \exp(-e^x)$

CUMULATIVELAPLACE $y = \frac{1}{2} e^x$ if $x < 0$
 $y = 1 - \frac{1}{2} e^{-x}$ otherwise

ELLIOTT	$y = \frac{x}{1 + x + 1 }$
EXPONENTIAL	$y = e^x$
EXPONENTIALLINEAR	$y = x \text{ if } x \geq 0$ $y = e^x - 1 \text{ otherwise}$
GAUSSIAN	$y = \exp(-x^2)$
GOMPERTZ	$y = e^x \exp(-x^2)$
HEAVISIDE	$y = 1 \text{ if } x > 0$ $y = 0 \text{ otherwise}$
LAPLACE	$y = e^{- x }$
LEAKYRECTIFIEDLINEAR	$y = x \text{ if } x > 0$ $y = 0.01x \text{ otherwise}$
LINEAR	$y = x$
LOGISTIC	$y = \frac{1}{1 + \exp(-x)}$
LOGISTICPRECISE	$y = \frac{1}{1 + \exp(-x)}$ Calculated to a higher precision than the LOGISTIC activation function.
RAMP	$y = 0 \text{ if } x \leq 0$ $y = x \text{ if } 0 < x < 1$ $y = 1 \text{ otherwise}$
RECIPROCALLINEAR	$y = \frac{1}{x} \text{ if } x > 1$ $y = 2 - x \text{ otherwise}$
RECTIFIEDLINEAR	$y = x \text{ if } x > 0$ $y = 0 \text{ otherwise}$
SIN	$y = \sin(x)$
SINC	$y = \frac{\sin(x)}{x}$

SOFTABS

$$y = x \left[\frac{1 - \exp(x)}{1 + \exp(x)} \right]$$

SOFTMAX

$$y_i = \frac{\exp(x_i)}{\sum_{n=1}^N \exp(x_n)}$$

where N is the number of network outputs.

SOFTPLUS

$$y = \log|1 + e^x|$$

SOFTSIGN

$$y = \frac{x}{1 + |x|}$$

SOFTSTEP

$$\begin{aligned} y &= 0 \text{ if } x \leq 0 \\ y &= x^2(3 - 2x) \text{ if } 0 < x < 1 \\ y &= 1 \text{ otherwise} \end{aligned}$$

SQUAREABS

$$\begin{aligned} y &= |x|^2 \text{ if } |x| \leq 1 \\ y &= 2|x| - 1 \text{ otherwise} \end{aligned}$$

SQUARE

$$y = x^2$$

TANH

$$y = \tanh(x)$$

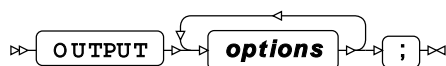
TANHPRECISE

$$y = \tanh(x)$$

Calculated to a higher precision than the `TANH` activation function.

OUTPUT

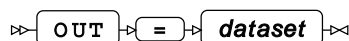
Creates an output dataset containing the input observations and the MLP outputs.



Options

The following *options* are available:

OUT



Specifies the name of the output dataset.

If `OUT` is not specified, the procedure creates the dataset as `DATAN` in the `WORK` library, where `n` is incremented for each output dataset.

OUTPUHIDDENACTIVITIES

➤ OUTPUTHIDDENACTIVITIES ➤

Specifies that the activities of hidden neurons are included.

OUTPUTINPUTACTIVITIES

➤ OUTPUTINPUTACTIVITIES ➤

Specifies that the activities of input neurons are included.

OPTIMALBIN procedure

The OPTIMALBIN procedure enables you to perform optimal binning on an input dataset.

About optimal binning

Optimal binning allows you to group observations into *partitions* (bins) according to the value of an *input* (predictor) variable in a way that the grouped values are still good predictors of the *target* (or response) variable.

For a discrete variable, the partitions are subsets of the possible values. For a continuous variable, the partitions are contiguous ranges of values.

Many data mining techniques become inefficient when one or more predictor variables have a large number of possible values. Optimal binning enables you to partition these variables into fewer groups in a way that maintains the *predictive power* of the original data.

`PROC OPTIMALBIN` uses algorithms developed by Raymond Anderson for partitioning the values of an input variable (Anderson, R. The Credit Scoring Toolkit, Oxford Press, 2007) and measures of *predictive power* developed by Mamdouh Reefat (Reefat, M. Credit Risk Scorecards: Development and Implementation Using SAS, Lulu.com, 2016).

The binning process for each variable works in two phases:

1. Fine classing. In this phase, significant values are grouped together into bins which are as evenly-sized as possible. No predictive power measure is used.

2. Coarse classing. In this phase, an iterative process uses the predictive power measure to determine the best pair of bins to merge together in each iteration. The process terminates when the specified stopping criteria apply.

`PROC OPTIMALBIN` allows you to specify one or more predictor variables in the input dataset, the target variable that you want to predict and the options that control the binning strategy to use. The optimal binning for each input variable is calculated independently of the other input variables. For each input variable, you can specify the number of bins to create in the fine classing phase and the maximum number of bins allowed in the coarse classing phase. You can also specify the predictive power measure to use and the various stopping criteria values (for example, when the minimum bin size is reached, or when the further grouping on that variable would not significantly improve the predictive power of the data).

The output shows, among other things, the optimal partitioning for each input variable, and the predictive power of the dataset if the observations were grouped in that way using that variable. You can examine the results to determine the best binning strategy to use on your data.

Predictive power criteria

Predictive power is a way of measuring how well a particular input variable can predict the target variable.

Pearson's Chi-Squared statistic

Pearson's Chi-squared statistic is a measure of the likelihood that the value of the target variable is related to the value of the predictor variable.

Each observation in the dataset is allocated to a cell in a contingency table, according to the values of the predictor and target variables. Pearson's Chi-squared statistic is calculated as the normalised sum of the squared deviations between the actual number of observations in each cell, and the expected number of observations in each cell if there were no relationship between the predictor and target variables.

If a predictor variable has a high Pearson's Chi-squared statistic, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Pearson's Chi-squared statistic for a discrete target variable is calculated as

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}}$$

where:

- N is the total number of observations in the dataset

- r is the number of distinct values of the predictor variable X (these are the rows in the contingency table)
- c is the number of distinct, discrete values of the target variable Y (these are the columns in the contingency table)
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j (these are the values in the cells of the contingency table)
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- μ_{ij} is the expected value of n_{ij} , calculated as

$$\mu_{ij} = \frac{n_{i*}n_{*j}}{N}$$

Entropy Variance

Entropy variance is a measure of how well the value of a predictor variable can predict the value of the target variable.

If a variable in a dataset has a high entropy variance, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Entropy variance for a discrete target variable is calculated as

$$E_r = 1 - \frac{\sum_{i=1}^r \left(\frac{n_{i*}E_i}{N} \right)}{E}$$

where:

- N is the total number of observations in the dataset
- r is the number of distinct values of the predictor variable, X
- c is the number of distinct, discrete values of the target variable, Y
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- E_i is the entropy calculated for just the observations where the predictor variable is X_i , calculated as

$$E_i = -\frac{1}{\log(c)} \sum_{j=1}^c \frac{n_{ij}}{n_{i*}} \log\left(\frac{n_{ij}}{n_{i*}}\right)$$

- E is the entropy calculated for all the observations, calculated as

$$E = -\frac{1}{\log(c)} \sum_{j=1}^c \frac{n_{*j}}{N} \log\left(\frac{n_{*j}}{N}\right)$$

Gini Variance

Gini variance is a measure of how well the value of a predictor variable can predict the target variable.

If a variable in a dataset has a high Gini variance, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

Gini variance for a discrete target variable is calculated as

$$G_r = 1 - \frac{\sum_{i=1}^r \left(\frac{n_{i*} G_i}{N} \right)}{G}$$

where

- N is the total number of observations in the dataset
- r is the number of distinct values of the predictor variable, X
- c is the number of distinct, discrete values of the target variable, Y
- n_{ij} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the j th value, Y_j
- n_{i*} is the total number of observations for which the predictor variable X has the i th value, X_i
- n_{*j} is the total number of observations for which the target variable Y has the j th value, Y_j
- G_i is the Gini impurity calculated for just the observations where the predictor variable is X_i , calculated as

$$G_i = 1 - \frac{\sum_{j=1}^c n_{ij}^2}{n_{i*}^2}$$

- G is the Gini impurity calculated for all the observations, calculated as

$$G = 1 - \frac{\sum_{j=1}^c n_{*j}^2}{N^2}$$

Information value

Information value is a measure of the likelihood that the value of the target variable is related to the value of the predictor variable. The information value measure is only applicable for binary target variables (that is, target variables that can take one of exactly two values).

If a predictor variable has a high information value, it means that the variable is a good predictor of the target variable, and is likely to be a good candidate to use to split the data in a binning or tree-building algorithm.

The information value statistic is calculated as

$$IV = \sum_{i=1}^r \left(\frac{n_{i0}}{n_{*0}} - \frac{n_{i1}}{n_{*1}} \right) WOE_i$$

where:

- r is the number of distinct, discrete values of the predictor variable X (these are the rows in the contingency table)
- Y_0 and Y_1 are the two possible values of the binary target variable Y
- n_{i0} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the value, Y_0 (these are the values in the cells of the Y_0 column in the contingency table)
- n_{i1} is the number of observations for which the predictor variable X has the i th value, X_i , and the target variable Y has the value, Y_1 (these are the values in the cells of the Y_1 column in the contingency table)
- n_{*0} is the total number of observations for which the target variable Y has the value, Y_0
- n_{*1} is the total number of observations for which the target variable Y has the value, Y_1
- $\alpha \ll 1$ is the weight of evidence (WOE) adjustment, a small positive number to avoid infinite values when $n_{i0} = 0$ or $n_{i1} = 0$
- WOE_i is the WOE value for observations where the predictor variable is X_i , calculated as

$$WOE_i = \ln\left(\frac{n_{i0}}{n_{*0}} + \alpha\right) - \ln\left(\frac{n_{i1}}{n_{*1}} + \alpha\right)$$

Using the OPTIMALBIN procedure

This example shows how to use PROC OPTIMALBIN to determine the optimal binning strategy for a dataset.

This example uses a simple dataset containing a sample of 100 people and their age, salary, make of car, whether they own a dog and whether they have defaulted on a loan. In this simple example, the OPTIMALBIN procedure is used to determine the following information, for each of the variables, age, salary and car:

- how effective the variable is at predicting whether that person has defaulted on a loan (the predictive power of the variable)
- the best way to group values of that variable in the analysis to make the data simpler to handle but ensure the best predictive power (the optimal binning strategy).

Dataset

This example uses the dataset `loanData`, which contains the following observations:

Age	Salary	Car	Dog	Loan	Default
21	21325	Ford	1	0	
21	30154	Ford	0	0	
21	52389	Ford	0	1	
22	59703	Ford	0	0	
22	34264	Ford	0	1	
22	9720	.	0	0	
22	43123	Ford	0	0	
22	65111	Ford	0	0	
23	48437	Ford	0	0	
24	3748	.	0	0	
24	42226	Ford	0	1	
24	36632	Ford	0	1	
25	48310	Ford	1	0	
25	27238	Ford	1	0	
26	25927	Ford	0	0	
26	59457	Nissan	0	0	
26	39058	Nissan	0	0	
27	66886	Nissan	0	1	
28	62063	Nissan	1	0	
29	55120	Nissan	0	0	
30	67674	Nissan	0	0	
32	7598	Ford	0	0	
33	15708	Ford	1	0	
33	53192	Nissan	0	0	
33	44778	Nissan	1	0	
34	9123	Ford	0	0	
36	93027	Volvo	0	0	
36	53889	Volvo	0	0	
37	106263	Volvo	0	0	
41	44477	Volvo	0	1	
41	34316	Nissan	1	1	
42	68092	Volvo	1	0	
42	59812	Volvo	0	0	
42	109801	Volvo	0	1	
43	67401	Volvo	0	0	
43	119848	Volvo	0	0	
43	29937	Ford	0	0	
43	83910	Volvo	0	0	
44	69805	Volvo	0	0	
44	10185	Ford	0	0	
44	70349	Volvo	0	1	
45	108497	Volvo	1	0	
45	18964	Ford	1	0	
45	63852	Volvo	0	0	
45	60078	Volvo	1	0	
46	110470	VW	0	0	
46	94727	VW	0	0	
46	120335	VW	0	0	
47	135657	VW	0	1	
47	117581	VW	0	0	
47	124175	VW	0	0	
47	21844	Ford	0	1	
47	23676	Ford	0	1	
48	80039	VW	0	0	
48	49712	Volvo	0	0	

49	42996	Volvo	1	0
50	24927	Ford	0	0
50	143032	VW	1	0
50	3377	Ford	0	0
51	100965	BMW	0	1
52	109383	BMW	0	0
52	152770	BMW	1	0
52	101555	BMW	0	0
52	95710	VW	0	0
52	147672	BMW	1	1
54	52246	Volvo	0	0
54	81418	VW	0	0
54	22060	Ford	0	1
54	115577	BMW	0	0
55	59722	Volvo	0	1
56	164395	BMW	0	0
56	157190	BMW	0	1
56	17268	Ford	0	0
56	60255	Volvo	0	0
56	162720	BMW	0	0
58	163492	BMW	0	0
59	114415	BMW	0	1
59	134879	BMW	0	0
59	33462	Nissan	0	1
60	7823	Ford	0	0
60	134460	BMW	0	0
61	773	Ford	0	0
64	30254	Nissan	0	0
64	154860	BMW	0	0
64	58275	Volvo	0	0
64	122884	BMW	0	0
65	138848	BMW	0	0
65	123111	BMW	1	0
66	147121	BMW	0	0
66	188638	BMW	0	0
66	165768	BMW	1	0
66	89132	VW	0	1
66	144425	BMW	0	0
68	91219	VW	0	0
68	193010	BMW	0	0
68	39799	Nissan	1	0
68	41234	Volvo	0	0
69	33319	Nissan	1	1
69	180753	BMW	1	0
70	16219	Ford	1	0

Code example

The model defines `age`, `salary`, and `car` as input variables. `age` is an ordinal variable (it contains discrete values with an implicit ordering), `salary` is an interval variable (it can be regarded as continuous) and `car` is a nominal variable (it contains discrete values with no implicit ordering). The variable that the binning strategy needs to predict is `loan_default`. The binning strategy is `GINIVAR` (the predictive power is measured using *Gini Variance*). The output shows the predictive power of each variable and the optimal bins that should be used to group values of this variable, if required.

```
PROC OPTIMALBIN
  DATA = loanData
  OUTSTATS = optbinstats_loanData
  CRITERION = GINIVAR
  OUTPUT = optbinout_loanData;
  INPUT age /LEVEL=ORDINAL;
  INPUT salary /LEVEL=INTERVAL;
  INPUT car /LEVEL=NOMINAL;
  TARGET loan_default / LEVEL=BINARY;
RUN;
```

Target Summary

The Target Summary table shows the target type that the optimal binning strategy is required to predict. In this case, it's a discrete target with binary (yes or no) values.

Target Summary				
Discrete or continuous? Descending?	Level type	Number of categories	Order type	
DISCRETE	BINARY	2	INTERNAL	NO

Input Summary

The Input Summary table summarises the input variables, their types, categories and the values that control the number of bins to try in the optimal binning algorithm.

Run Summary

The Run Summary table summarises information about the run, for example, the input dataset, the binning criterion chosen, and the number of data items processed.

Performance

The Performance table lists each of the predictor variables, the number of bins each should be split into for optimal predicative power and the value of the predictive power for this split. In this example, the variable with the highest predictive power is `salary`, so, if the data in this dataset is grouped, `salary` is the grouped variable that is most likely to be able to predict, on its own, whether someone will default

on a loan. Note that this procedure does not actually group the data or derive a model to use salary to predict the likelihood of someone defaulting of their loan: it simply indicates how the data should best be grouped to avoid losing information, and if it were grouped, how effective any predictions made from it are likely to be.

Output dataset

The output dataset is a table that shows the optimal bin split for each variable. For ordinal and nominal variables (age and make of car in this example) there is an observation for each discrete value that appears in the input dataset, and a bin number for each. For example, the optimal binning for the age variable would use 10 bins, and put ages 21, 22, 23 and 24 in bin 1, ages 25, 26 and 27 in bin 2 and so on. The optimal binning for the make of car variable would put undefined values, BMW, and Volvo in bin 1, Nissan in bin 2 and Ford in bin 3.

For each continuous variable, for each of the bins in the optimal split, there is an entry for the lower bound of the bin and an entry for the upper bound. For example, if the salary variable were used for optimal binning, the lower bound of bin 1 is minus infinity, and the upper bound is £18,964.50. The lower bound of bin 2 is £18,964.50 (the same as the upper bound of bin 1) and the upper bound of bin 2 is £30,254.01

NAME	_BINIDX_	_TYPE_	VAR1	VAR2	VAR3
AGE	1	ORDINAL	21		.
AGE	1	ORDINAL	22		.
AGE	1	ORDINAL	23		.
AGE	1	ORDINAL	24		.
AGE	2	ORDINAL	25		.
AGE	2	ORDINAL	26		.
AGE	2	ORDINAL	27		.
AGE	3	ORDINAL	28		.
AGE	3	ORDINAL	29		.
AGE	3	ORDINAL	30		.
AGE	3	ORDINAL	32		.
AGE	3	ORDINAL	33		.
AGE	3	ORDINAL	34		.
AGE	3	ORDINAL	36		.
AGE	3	ORDINAL	37		.
AGE	4	ORDINAL	41		.
AGE	4	ORDINAL	42		.
AGE	4	ORDINAL	43		.
AGE	4	ORDINAL	44		.
AGE	5	ORDINAL	45		.
AGE	5	ORDINAL	46		.
AGE	6	ORDINAL	47		.
AGE	7	ORDINAL	48		.
AGE	7	ORDINAL	49		.
AGE	7	ORDINAL	50		.
AGE	8	ORDINAL	51		.
AGE	8	ORDINAL	52		.
AGE	8	ORDINAL	54		.
AGE	9	ORDINAL	55		.
AGE	9	ORDINAL	56		.
AGE	9	ORDINAL	58		.
AGE	9	ORDINAL	59		.
AGE	10	ORDINAL	60		.

AGE	10	ORDINAL	61	.
AGE	10	ORDINAL	64	.
AGE	10	ORDINAL	65	.
AGE	10	ORDINAL	66	.
AGE	10	ORDINAL	68	.
AGE	10	ORDINAL	69	.
AGE	10	ORDINAL	70	.
MAKE_OF_CAR	1	NOMINAL	.	.
MAKE_OF_CAR	1	NOMINAL	BMW	.
MAKE_OF_CAR	1	NOMINAL	VW	.
MAKE_OF_CAR	1	NOMINAL	Volvo	.
MAKE_OF_CAR	2	NOMINAL	Nissan	.
MAKE_OF_CAR	3	NOMINAL	Ford	.
SALARY	1	LOWER_BOUND		-infty
SALARY	1	UPPER_BOUND		18963.50
SALARY	2	LOWER_BOUND		18963.50
SALARY	2	UPPER_BOUND		30254.01
SALARY	3	LOWER_BOUND		30254.01
SALARY	3	UPPER_BOUND		44477.39
SALARY	4	LOWER_BOUND		44477.39
SALARY	4	UPPER_BOUND		52389.30
SALARY	5	LOWER_BOUND		52389.30
SALARY	5	UPPER_BOUND		59703.34
SALARY	6	LOWER_BOUND		59703.34
SALARY	6	UPPER_BOUND		115576.81
SALARY	7	LOWER_BOUND		115576.81
SALARY	7	UPPER_BOUND		134879.20
SALARY	8	LOWER_BOUND		134879.20
SALARY	8	UPPER_BOUND		162720.37
SALARY	9	LOWER_BOUND		162720.37
SALARY	9	UPPER_BOUND		infty

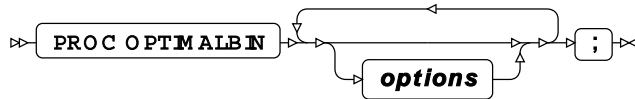
OPTIMALBIN procedure reference

Describes the syntax and options for PROC OPTIMALBIN and its contained statements.

PROC OPTIMALBIN ↗	3306
Determines the optimal way to split a dataset into groups, while retaining the predictive power of the dataset.	
BY ↗	3309
Groups the observations in a dataset using the specified variables.	
FREQ ↗	3310
Specifies a variable containing the frequency associated with an observation.	
INPUT ↗	3310
Specifies the input variables to which measures of predictive power and the optimal binning strategy are applied.	
TARGET ↗	3312
Specifies the target (dependent) variable and any options that apply to the variable.	
WHERE ↗	3314
Restricts the observations to be processed.	

PROC OPTIMALBIN

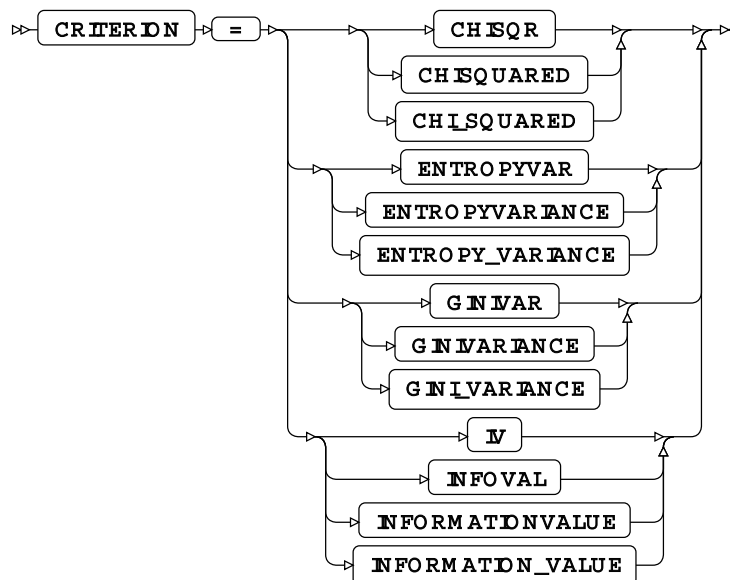
Determines the optimal way to split a dataset into groups, while retaining the predictive power of the dataset.



Options

The following *options* are available:

CRITERION



Specifies the split criterion used when performing optimal binning. The **CRITERION** specified determines the way that predictive power is measured which in turn is used to determine the optimal split strategy.

This option is mandatory and must be specified.

CHISQR

Specifies that *Pearson's Chi-Squared Test* is used to measure the predictive power of variables. This option cannot be used with continuous target variables (that is, the **TARGET** statement must not specify `/LEVEL=INTERVAL`).

ENTROPYVAR

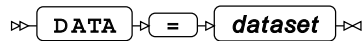
Specifies that *Entropy Variance* is used to measure the predictive power of variables.

GINIVAR

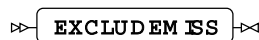
Specifies that *Gini Variance* is used to measure the predictive power of variables by measuring the strength of association between variables.

IV

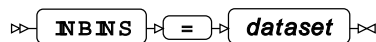
Specifies that *Information Value* is used to measure the predictive power of variables. This option can only be used with binary target variables (the `TARGET` statement specifies `LEVEL=BINARY`).

DATA

Specifies the input dataset. If an input dataset is not specified, the most recently-created dataset is used.

EXCLUDEMISS

Specifies that observations containing missing values are excluded when creating bins.

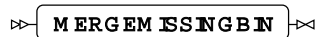
INBINS

Specifies a dataset containing pre-calculated or specified bins from which the weight of evidence (WoE) associated with a specified target variable is calculated.

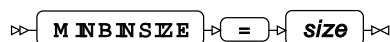
If specified, the values in this dataset override the values computed directly from the specified input file.

MAXPREDICTIVEPOWERCHANGE

Specifies the maximum change allowed in the predictive power when optimally merging bins.

MERGEMISSINGBIN

Specifies that missing values are considered a separate valid category when binning data.

MINBINSIZE

Specifies the minimum number of observations in each bin. If a proposed split would create a bin containing fewer than the specified minimum number of observations, the split does not occur. The default value is 2.

MINBINSIZERATIO

» **MINBINSIZERATIO** = *ratio* «

Specifies the minimum size of each bin as a proportion of the number of input observations, expressed as a percentage. If a proposed split would create a bin containing fewer than the specified percentage of observations, the split does not occur. The default value is 5 (percent).

MONOTONEWOE

» **MONOTONEWOE** «

Ensures that the weight of evidence (WoE) value for ordered input variables is either monotonically increasing or monotonically decreasing.

NOOPENLEFT

» **NOOPENLEFT** «

Specifies that when binning a continuous variable, the bin containing the very lowest values (the bin on the far left) has a closed lower bound. Otherwise, the lower bound of the bin containing the very lowest values is $-\infty$ (minus infinity).

NOOPENRIGHT

» **NOOPENRIGHT** «

Specifies that when binning a continuous variable, the bin containing the very highest values (the bin on the far right) has a closed upper bound. Otherwise, the upper bound of the bin containing the very highest values is ∞ (infinity).

NOPRINT

» **NOPRINT** «

Specifies that no ODS output is created.

OUTPUT

» **OUTPUT** = *dataset* «

Specifies the output dataset that contains the optimal bin split.

OUTSTATS

» **OUTSTATS** = *dataset* «

Specifies the dataset that contains the output the summary statistics (the measures of predicted power).

OUTSTATSONLY

➤ OUTSTATSONLY ➤

Specifies that only the measures of predicted power are calculated. No optimal binning of the input variables takes place.

WOE

➤ WOE ➤

Specifies that the weight of evidence (WoE) table is calculated and added to the printed output. This table contains the weight of evidence and information values associated with each of the generated bins for the target variable. This option is ignored if `OUTPUTSTATSONLY` is specified.

WOEADJUST

➤ WOEADJUST ➤ = ➤ *adjustment* ➤

Specifies the adjustment applied in weight of evidence calculations to avoid invalid results for pure inputs. This option is only valid if `CRITERION=IV`.

The default value of `WOEADJUST` is `1E-5`.

BY

Groups the observations in a dataset using the specified variables.

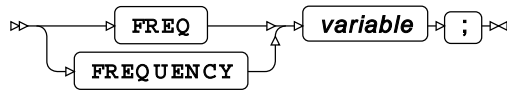
➤ BY ➤ *variable* ➤ ; ➤

The specified variable or variables are used to separate the input data into groups. `PROC OPTIMALBIN` generates a separate model from the data in each group.

If the `BY` statement is included, the input dataset must be presorted on the specified variable or variables. If a variable is specified as an input variable in the `INPUT` statement, or as a target variable in `TARGET` statement, it cannot also be specified in the `BY` statement.

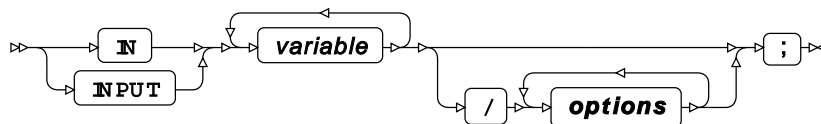
FREQ

Specifies a variable containing the frequency associated with an observation.



INPUT

Specifies the input variables to which measures of predictive power and the optimal binning strategy are applied.



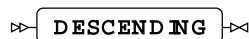
variable

A variable to which measures of predictive power and optimal binning are applied.

Options

The following *options* are available:

DESCENDING



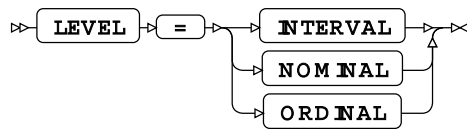
Specifies a descending sort order for the variable. This option only applies if `ORDER=FORMATTED` or `ORDER=INTERNAL` is also specified, as the sort order cannot be explicitly determined from these options. If not specified, the variable is assumed to be in ascending sort order.

If `ORDER` is not equal to `FORMATTED` or `INTERNAL` then the sort order for the variable is determined by the `ORDER` option.

INITNUMBINS



Specifies the initial number of bins to use. The default initial number of bins is 50. The initial number of bins must be between 2 and 1000.

LEVEL

Specifies the level for the input variables. The default **LEVEL** value is **INTERVAL**.

INTERVAL

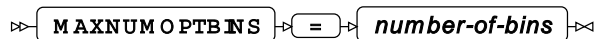
Specifies a continuous input variable with an implicit category ordering.

NOMINAL

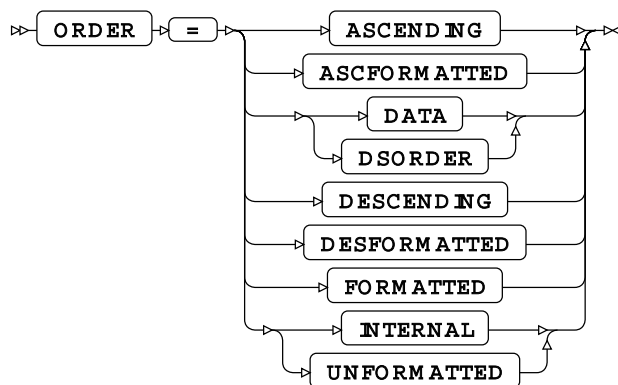
Specifies a discrete input variable with no implicit ordering. When defining the optimal bins, any categories can be merged together to generate the strategy.

ORDINAL

Specifies a discrete input variable with an implicit category ordering. When optimally binning, only adjacent categories can be merged together to generate the strategy.

MAXNUMOPTBINS

Specifies the maximum number of optimal bins to use. The default maximum number of bins is 10. The specified number must be between 2 and 100.

ORDER

Specifies the order of input variables. The default **ORDER** value is **ASCENDING**.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

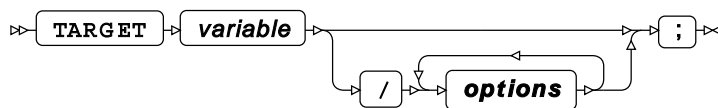
The variable is sorted by the formatted value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

TARGET

Specifies the target (dependent) variable and any options that apply to the variable.

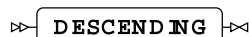


Only one `TARGET` statement is allowed in each `PROC OPTIMALBIN` statement. The `TARGET` statement must contain a single variable.

Options

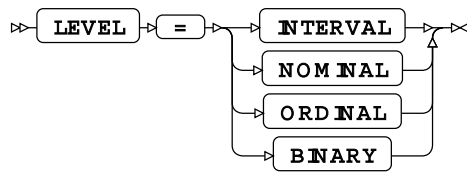
The following *options* are available:

DESCENDING



Specifies a descending sort order for the variable. This option only applies if `ORDER=FORMATTED` or `ORDER=INTERNAL` is also specified, as the sort order cannot be explicitly determined from these options. If not specified, the variable is assumed to be in ascending sort order.

If `ORDER` is not equal to `FORMATTED` or `INTERNAL` then the sort order for the variable is determined by the `ORDER` option.

LEVEL

Specifies the level for the target variable. The default **LEVEL** value is **INTERVAL**.

INTERVAL

Specifies a continuous target variable containing an implicit category ordering.

NOMINAL

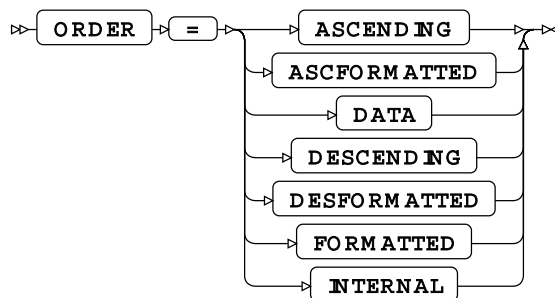
Specifies a discrete target variable with no implicit ordering.

ORDINAL

Specifies a discrete target variable with an implicit category ordering.

BINARY

Specifies a target variable that can take one of two values.

ORDER

Specifies the order of the target variable. The default **ORDER** value is **INTERNAL**.

ASCENDING

The variable is sorted by ascending order of the raw value.

ASCFORMATTED

The variable is sorted by ascending order of the formatted value.

DATA

The variable is sorted by the order in which the values of the variable first occur when the data is read.

DESCENDING

The variable is sorted by descending order of the raw value.

DESFORMATTED

The variable is sorted by descending order of the formatted value.

FORMATTED

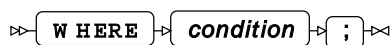
The variable is sorted by the formatted value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

INTERNAL

The variable is sorted by the raw value. If the (separate) `DESCENDING` option is also specified, the sort order is descending, otherwise, the sort order is ascending.

WHERE

Restricts the observations to be processed.



SEGMENT procedure

The `SEGMENT` procedure enables you to segment an input dataset to highlight similarities and differences in the data.

About segmentation

Segmentation is a way of separating the observations in a dataset into a number of groups (or segments) where the observations in each group tend to have similar characteristics and observations in different groups tend to have different characteristics.

Segmentation allows you to visualise the common characteristics of the observations in each segment, and see how they differ from the observations in the other segments.

The method is derived indirectly from a voting system proposed in the seventeenth century by a French mathematician Condorcet. He suggested that voters should order all candidates by preference. These orderings, he suggested, should then be converted to a series of pairs of the form: candidate x is preferred to candidate y. The elected candidate would be the one who is preferred by the majority of such pairings aggregated over all voters. It was not a very practical voting method before the age of computers, but more recently, Pierre Michaud used the principle to segment literary texts according to various properties of the texts. The method described here is based on that described by Michaud (Michaud, 1995 [1]).

Although the method does not have a strong underlying statistical foundation, the results are intuitively appealing to marketers. It can be applied, for example, to subdivide large customer databases into smaller groups with similar characteristics so products and marketing literature can be targeted at particular market segments. The method has also found a useful niche for characterising groups who have a high propensity to take particular actions, such as lapsing insurance policies.

In a perfect segmentation:

- all observations in the same segment have the same value for all variables specified for the segmentation
- two observations in different segments have different values for each variable specified in the segmentation.

This can be achieved for a segmentation on a single variable by creating one segment for each possible value of that variable. But in practice, a segmentation will be based on more than one variable. Then, if one variable is segmented perfectly, another variable is likely to have different values in the same segment and the same value in different segments. For example, if all observations where `sex=male` are in one segment and all observations where `sex=female` are in another segment, some observations in the `male` segment may have `eyes=blue` and some may have `eyes=brown`. You could separate the observations into further segments based on eye colour (for example, `sex=male, eyes=blue` in one segment, and `sex=male, eyes=brown` in another) but now all observations where `sex=male` are no longer in the same segment.

`PROC SEGMENT` attempts to maximise the similarities between observations in a segment and maximise the differences between observations in different segments. For each variable in each pair of observations, `PROC SEGMENT` calculates the degree of similarity between the two values of the variable, then uses an iterative process to optimally allocate observations to segments. Initially, observations are assigned to segments speculatively, then iteratively refined until there is no further improvement in the similarity and difference scores.

Segmentation calculations

The segmentation calculations maximise the similarity between the observations in a segment and maximise the differences between observations in different segments.

Suppose there are n observations, \mathbf{X}_1 to \mathbf{X}_n , with m variables, where observation \mathbf{X}_i has values $(X_{i1}, X_{i2}, X_{i3}, \dots, X_{im})$ and observation \mathbf{X}_j has values $(X_{j1}, X_{j2}, X_{j3}, \dots, X_{jm})$.

The optimal segmentation allocates the observations to G segments, \mathbf{S}_1 to \mathbf{S}_G for some value of G to be determined, such that the similarity is maximised between values of variables in observations in the same segment, and the difference is maximised between values of variables in different segments. The function to maximise is calculated as:

$$\sum_{g=1}^G \sum_{i=1}^n \sum_{j=i+1}^n \sum_{k=1}^m W_k(X_{ik}, X_{jk}) [I(\mathbf{X}_i, \mathbf{X}_j \in \mathbf{S}_g) (1 - \Delta_k(X_{ik}, X_{jk})) + I(\mathbf{X}_i \in \mathbf{S}_g, \mathbf{X}_j \notin \mathbf{S}_g) \Delta_k(X_{ik}, X_{jk})]$$

where

- $W_k(X_{ik}, X_{jk})$ is a weighting factor for the k^{th} variable, calculated for X_{ik} and X_{jk} . This value is a combination of a variable-specific weighting constant for the k^{th} variable and a value weighting function that specifies the importance to attach to a particular pair of values, relative to other pairs of values of values. For more information see *Weighting factor* [↗](#) (page 3320).
- $I(\mathbf{X}_i, \mathbf{X}_j \in \mathbf{S}_g)$ is an indicator function with the value 1 if both observation \mathbf{X}_i and observation \mathbf{X}_j are in segment \mathbf{S}_g and value 0 (zero) otherwise.
- $I(\mathbf{X}_i \in \mathbf{S}_g, \mathbf{X}_j \notin \mathbf{S}_g)$ is an indicator function with the value 1 if observation \mathbf{X}_i is in segment \mathbf{S}_g and observation \mathbf{X}_j is not in segment \mathbf{S}_g and value 0 (zero) otherwise.
- $\Delta_k(X_{ik}, X_{jk})$ is the value of the difference function for k^{th} variable, calculated for the values X_{ik} and X_{jk} . This function returns a value between 0 and 1 to indicate how different two values of a variable are. A value of 0 (zero) indicates that the values are considered completely similar, and a value of 1 indicates that the values are considered completely different. For more information about the difference functions see *Difference functions* [↗](#) (page 3316).

The function above maximises the similarity, $1 - \Delta_k(X_{ik}, X_{jk})$, for observations in the same segment (when $I(\mathbf{X}_i, \mathbf{X}_j \in \mathbf{S}_g)$) and maximises the differences, $\Delta_k(X_{ik}, X_{jk})$, for observations in different segments (when $I(\mathbf{X}_i \in \mathbf{S}_g, \mathbf{X}_j \notin \mathbf{S}_g)$).

Difference functions

Difference functions measure the degree of difference or similarity between two values of a variable.

Comparing variable values

PROC SEGMENT provides several difference functions for comparing values of variables. The COMPARE option on the INPUT statement determines the difference function that is used.

The COMPARE option also affects whether values of the variable are regarded as being from a discrete set or a continuous range.

Depending on the COMPARE option chosen, the measure of the degree of difference may be binary (the only possible values are 0 or 1), discrete (there are multiple possible distinct values between 0 and 1) or continuous (any value between 0 and 1 is possible). A value of 0 (zero) indicates that the difference function considers two values to be identical, and a value of 1 indicates that the difference function considers two values to be completely different.

PROC SEGMENT supports the following COMPARE options for discrete variables:

- NOMINAL, for discrete variables that have no explicit ordering (for example, marital status). See *NOMINAL comparisons* [↗](#) (page 3318).
- ORDINAL, for discrete variables that have an explicit ordering (for example, number of siblings). See *ORDINAL comparisons* [↗](#) (page 3318).

- **BANDED**, for variables that have been grouped into a number of discrete, ordered bands (for example, age ranges). See *BANDED comparisons* [↗](#) (page 3317).

PROC SEGMENT supports the following COMPARE options for continuous variables:

- **ABSOLUTE**. See *ABSOLUTE comparisons* [↗](#) (page 3317).
- **RANK**. See *RANK comparisons* [↗](#) (page 3318).
- **RELATIVE**. See *RELATIVE comparisons* [↗](#) (page 3319).

ABSOLUTE comparisons

The COMPARE=ABSOLUTE (low-difference, high-difference) option specifies that the variable is treated as a continuous variable.

The difference between two values is a function of the absolute difference between the two values. *low-difference* specifies the absolute difference below which two values are considered identical, with a difference value of 0 (zero). *high-difference* specifies the absolute difference above which two values are considered completely different, with a difference value of 1. Pairs of values with absolute differences between *low-difference* and *high-difference* have difference values calculated linearly between 0 and 1.

The difference function $\Delta(X_i, X_j)$ is:

$$\begin{aligned} & 0 \quad \text{if } |X_i - X_j| < \text{low} \\ & 1 \quad \text{if } |X_i - X_j| \geq \text{high} \\ & 0.5 \quad \text{if } |X_i - X_j| = \text{low} \text{ and } (\text{low} = \text{high}) \\ & \frac{|X_i - X_j| - \text{low}}{\text{high} - \text{low}} \quad \text{otherwise} \end{aligned}$$

where

- *low* is the specified *low-difference*
- *high* is the specified *high-difference*.

BANDED comparisons

The COMPARE=BANDED option specifies that the variable is treated as a discrete variable.

The difference between two values is measured by the estimated proportion of the population between the two values. Two values in the same band have a difference value of slightly greater than 0. Two values, where one is in the lowest band and the other is in the highest band, have a difference value of slightly less than 1. This reflects that fact that the underlying values are likely to be slightly different, although they are in the same band.

The difference function $\Delta(X_i, X_j)$ is:

$$\begin{aligned} & \frac{1}{3}p_i \quad \text{if } X_i = X_j \\ & \left(\frac{1}{2}p_i + \frac{1}{2}p_j + \sum_{k>i}^{k<j} p_k \right) \quad \text{otherwise} \end{aligned}$$

where:

- p_i is the proportion of the population with value X_i
- p_j is the proportion of the population with value X_j
- $i \leq j$.

NOMINAL comparisons

The `COMPARE=NOMINAL` option specifies that the variable is treated as a discrete variable with no implicit ordering.

Two values are considered identical, with a difference value of 0 (zero), if they are identical. Otherwise they are considered completely different, with a difference value of 1.

The difference function $\Delta(X_i, X_j)$ is:

$$\begin{array}{ll} 0 & \text{if } X_i = X_j \\ 1 & \text{if } X_i \neq X_j \end{array}$$

ORDINAL comparisons

The `COMPARE=ORDINAL` option specifies that the variable is treated as a discrete variable with an implicit ordering.

Two values are considered identical, with a difference value of 0 (zero), if they are identical. Two values are considered completely different, with a difference value of 1, if one is in the lowest category and one is in the highest category. Otherwise the difference is measured by the fraction of the number of categories between them.

The difference function $\Delta(X_i, X_j)$ is:

$$\frac{|R_i - R_j|}{m - 1}$$

where:

- m is the number of different values of the variable X
- R_i is the position of the value of X_i in the ordered list of categories
- R_j is the position of the value of X_j in the ordered list of categories.

RANK comparisons

The `COMPARE=RANK` (low-difference, high-difference) option specifies that the variable is treated as a continuous variable.

The difference between two values is measured by the proportion of the population between the two values. *low-difference* specifies the proportion of the population below which two values are considered identical, with a difference value of 0 (zero). *high-difference* specifies the proportion of the population above which two values are considered completely different, with a difference value of 1. Pairs of values with differences between *low-difference* and *high-difference* have difference values calculated linearly between 0 and 1.

The difference function $\Delta(X_i, X_j)$ is:

$$\begin{aligned} & 0 \quad \text{if } |P(X_i) - P(X_j)| < \text{low} \\ & 1 \quad \text{if } |P(X_i) - P(X_j)| \geq \text{high} \\ & 0.5 \quad \text{if } |P(X_i) - P(X_j)| = \text{low} \quad \text{and} \quad (\text{low} = \text{high}) \\ & \frac{|P(X_i) - P(X_j)| - \text{low}}{\text{high} - \text{low}} \quad \text{otherwise} \end{aligned}$$

where:

- $P(X_i)$ is the proportion of the population less than X_i
- $P(X_j)$ is the proportion of the population less than X_j
- *low* is the specified *low-difference*
- *high* is the specified *high-difference*.

RELATIVE comparisons

The `COMPARE=RELATIVE(base-value, epsilon)` option specifies that the variable is treated as a continuous variable.

The difference between two values is determined by comparing their absolute difference with a value based on the root mean square of the two values. If the absolute difference is less than the root mean square value, the two values are considered identical, with a difference value of value of 0 (zero). If the absolute difference is equal to the root mean square value, the difference value is 0.5. Otherwise, the two values are considered completely different with a difference value of 1. *base-value* specifies an offset to apply to each value before calculating the root mean square, and *epsilon* specifies a scale factor to apply to the calculated root mean square value before comparing it with the absolute value.

The difference function $\Delta(X_i, X_j)$ is:

$$\begin{aligned} & 0 \quad \text{if } |X_i - X_j| < \varepsilon \sqrt{\frac{1}{2}((X_i - \text{base})^2 + (X_j - \text{base})^2)} \\ & 0.5 \quad \text{if } |X_i - X_j| = \varepsilon \sqrt{\frac{1}{2}((X_i - \text{base})^2 + (X_j - \text{base})^2)} \\ & 1 \quad \text{if } |X_i - X_j| > \varepsilon \sqrt{\frac{1}{2}((X_i - \text{base})^2 + (X_j - \text{base})^2)} \end{aligned}$$

where ε is the specified *epsilon* and *base* is the specified *base-value*.

Weighting factor

The weighting factor specifies the importance to attach to similarities and differences between values of a variable in pairs of observations.

The weighting factor $W_k(X_{ik}, X_{jk})$ is the product of a variable-specific weighting constant for the k^{th} variable and a value weighting function evaluated for X_{ik} and X_{jk} . That is:

$$W_k(X_{ik}, X_{jk}) = w_k W(X_{ik}, X_{jk})$$

where:

- w_k is the variable-specific weight constant for the k^{th} variable
- X_{ik} and X_{jk} are the values of the i^{th} and j^{th} variables respectively in the k^{th} observation
- $W(X_{ik}, X_{jk})$ is the value of the value weighting function for the values X_{ik} and X_{jk} .

Variable-specific weight constant

The variable-specific weight constant is specified by the `WEIGHT` option of the `INPUT` statement. The specified value is a numeric value that applies to a single variable. It specifies the degree of importance to attach to similarities and differences in the value of that variable in the overall segmentation.

Similarities and differences between values of a variable with a lower weight constant contribute less to the overall measure of similarity and difference than similarities and differences between values of a variable with a higher weight constant.

Value weighting function

The value weighting function is specified by the `WEIGHTING` option of the `PROC SEGMENT` statement. It specifies the degree of importance to attach to the similarity in or difference between a particular pair of values of a variable, relative to other pairs of values of that variable.

For example, if a particular value of a variable is very common, it may be appropriate to give a lower weight to the fact that it is the same in two observations. Whereas, if a particular value of a variable is rare, the fact that the variable has the same value in two observations may be more significant.

`PROC SEGMENT` supports the following value weighting functions:

- `CONSTANT` weighting, where all agreements between variable values in pairs of observations contribute an equal weight to the segmentation calculation.
- `LOG` weighting, where agreements between variable values in pairs of observations contribute a weight to the segmentation calculation that is proportional to the negative logarithm of the probability of that pairing occurring.
- `PROBABILITY` weighting, where agreements between variable values in pairs of observations contribute a weight to the segmentation calculation that is inversely proportional to the probability of that pairing occurring.

The LOG weighting option and the PROBABILITY weighting option are experimental.

CONSTANT weighting

All agreements between variable values in pairs of observations contribute an equal weight to the segmentation calculation. This means that overall, common pairings contribute more weight to the segmentation than uncommon pairings, because there are more of them.

The weighting function $W(X_{ik}, X_{jk})$ is 1 for all X_{ik} and X_{jk} .

LOG weighting

The values in all pairs of observations contribute a weight to the segmentation that is proportional to the negative logarithm of the probability of that pairing occurring. This means that the effect of pairings of values of a variable is proportional to the amount of information in that pairing.

The weighting function $W(X_{ik}, X_{jk})$ is:

$$-\log_2(p(X_{ik}, X_{jk}))$$

where

- X_{ik} and X_{jk} are the values of the i^{th} and j^{th} variables respectively in the k^{th} observation
- $p(X_{ik}, X_{jk})$ is the probability of the pair of values X_{ik} and X_{jk} occurring in the two randomly chosen observations from the dataset

If the number of observations, n , in the dataset is large enough, so that the difference between n and $n - 1$ can be ignored, $p(X_{ik}, X_{jk})$ can be calculated as:

$$\frac{n_{ik}n_{jk}}{n^2}$$

where n_{ik} is the number of observations in the dataset with the value X_{ik} , and n_{jk} is the number of observations in the dataset with the value X_{jk} .

PROBABILITY weighting

The values of the variables in all pairs of observations contribute a weight to the segmentation that is inversely proportional to the probability of that pairing occurring. This means that rare pairings of values of a variable have relatively more effect on the segmentation than common pairings, so overall, every pairing of variables contributes an equal amount. Each instance of an uncommon pair of values contributes more weight, but there are fewer of them. Similarly, each instance of a common pair of values contributes less weight, but there are more of them.

The weighting function $W(X_{ik}, X_{jk})$ is:

$$\frac{1}{p(X_{ik}, X_{jk})}$$

where

- X_{ik} and X_{jk} are the values of the i^{th} and j^{th} variables respectively in the k^{th} observation
- $p(X_{ik}, X_{jk})$ is the probability of the pair of values X_{ik} and X_{jk} occurring in the two randomly chosen observations from the dataset.

If the number of observations, n , in the dataset is large enough, so that the difference between n and $n - 1$ can be ignored, $p(X_{ik}, X_{jk})$ can be calculated as:

$$\frac{n_{ik}n_{jk}}{n^2}$$

where n_{ik} is the number of observations in the dataset with the value X_{ik} , and n_{jk} is the number of observations in the dataset with the value X_{jk} .

ODS Outputs

`PROC SEGMENT` ODS outputs include tables and plots.

By default, `PROC SEGMENT` produces a number of tables in ODS output which contain the segmentation options, the segmentation completion status and a summary of the final segmentation.

There are options to output additional tables such as the distribution of the values of each variable in each segment (option `DISTRIBUTIONS`) or the relative entropy of each variable in each segment (option `ENTROPY`).

The output includes a set of summary plots for each segment. By default, the plots show the proportion of the observations in the segment, the most significant variables in the segment, and the distribution of those variables for the observations in the segment, compared with the distribution of those variables in the population as a whole.

The most significant variables in a segment are those where the distribution of the values of the variable in the segment is most dissimilar to the distribution of the values of the variable in the overall population.

Scoring datasets

Once you have created a segmentation for a dataset, you can use it to score that dataset or another dataset.

The `SCORE` statement takes the data in the specified dataset and scores it using the segmentation defined in `PROC SEGMENT`. The score results are saved in a table which can be printed or saved in an output dataset. For each observation in the dataset being scored, the score results show the segment that the observation matches most closely, and the score (a measure of how similar that observation is to the other observations in the segment).

Saving and reusing segmentation models

Once you have produced a segmentation there are several ways you can save and reuse it.

You can use `PROC SEGMENT OUTMODEL` to save the segmentation together with its controlling parameters in a dataset. You can then use `PROC SEGMENT INMODEL` to score another dataset using the saved segmentation.

You can also use `PROC SEGMENT OUTEST` to save the segmentation without the controlling parameters in a dataset. You can then use `PROC SEGMENT INEST` to initialise the segmentation process for another similar dataset, or to continue refining the original segmentation.

The `OUTEST` option differs from the `OUTMODEL` option in that `OUTMODEL` saves the segmentation parameters, the distribution of the data used to create the segmentation and the segmentation of that data, whereas `OUTEST` saves the distribution of the data used to create the segmentation and the segmentation of that data, but not the segmentation parameters.

The saved output from `OUTMODEL` can be used directly to segment another dataset. When the saved output is used in a new segmentation, the `INPUT` statement options are read from the saved parameters.

When a dataset saved using `OUTEST` is used in a new segmentation, the `INPUT` statement options are derived from the current `INPUT` statements and not from the `INPUT` statement options that were specified when the `OUTEST` dataset was created. It is recommended that you specify the same `INPUT` statements, although you do not have to.

Using the SEGMENT procedure

This example shows how to use `PROC SEGMENT` segment a dataset.

Example

This example describes how the `SEGMENT` procedure can be applied to the publicly-available Iris dataset (Fisher, 1936 [4]).

The Iris dataset consists of measurements of the widths and lengths of the petals and sepals of three species of Iris. `PROC SEGMENT` is used to separate the observations into segments with similar sepal and petal widths and lengths. The output plots show how the distribution of measurements for the observations in each segment compare with the overall population distribution and how the observations from each species are allocated to each segments.

```
PROC SEGMENT DATA=iris DETAILS DIFFERENCES ENTROPY;  
INPUT sepal_length sepal_width petal_length petal_width/COMPARE=RANK BINS=8;  
INPUT species/REPORTONLY;  
OUTPUT OUT= iris_segment SEGMENT=segment_out SCORE=score_value;  
RUN;;
```

In this example:

- the four measurements, `sepal_length`, `sepal_width`, `petal_length` and `petal_width` are specified as continuous input variables, each of which are split into eight bins and are compared using `RANK` comparison
- the Iris species, `species`, is specified with the `REPORTONLY` option, which means that the variable is not included in the segmentation optimisation calculations, but is shown in the segmentation plots
- as well as the default output tables, options `DIFFERENCES`, `DETAILS`, and `ENTROPY` specify that the Differences, Details and Entropy tables are output
- the observations, the allocated segments and the score values are written to an output dataset, `iris_segment`, in the `WORK` library.

Segment Information

The `Segment Information` table records the dataset being segmented, and the general weighting option used.

In this example, no explicit `WEIGHTING` option was specified, so the default `CONSTANT` weighting was used.

Segmentation Information

Description	Value
Data Set	iris
Weighting	Constant

Segment Variable Information

The `Segment Variable Information` table lists the variables used in the segmentation, their types and the comparison methods chosen. In this example, the `species` variable has been included in the segmentation but is not active in the segmentation calculations.

Segment Variable Information

Variable	Active	Type	Equality Rule
<code>sepal_length</code>	Yes	Continuous	Rank
<code>sepal_width</code>	Yes	Continuous	Rank
<code>petal_length</code>	Yes	Continuous	Rank
<code>petal_width</code>	Yes	Continuous	Rank
<code>species</code>		Discrete	Nominal

Differences

The `Differences` tables show, for each input variable, the calculated differences between values of that variable in pairs of observations. There is a `Differences` table for each active input variable. In this example, all the input variables are continuous and each is split into eight bins. So the `Differences` tables show the calculated differences between values in each of the eight bins.

The `Differences` table for `petal_length` is shown here. The other `Differences` tables are similar.

In this Differences table:

- Pairs of observations with `petal_length` values in the same bin have a difference value of 0 (zero).
- Pairs of values in bins that are far apart have a difference value of 1. For example, a value in bin `2.5-< 3.25` and a value in bin `5.5-< 6.25` have a difference value of 1.
- Pairs of values in bins that are closer together have difference values between 0 and 1. For example, a value in bin `3.25-< 4` and a value in bin `4-< 4.75` have a difference value of 0.288.

The Differences tables are only shown if the `DIFFERENCES` option is specified.

Differences for petal_length								
petal_length	1-< 1.75	1.75-< 2.5	2.5-< 3.25	3.25-< 4	4-< 4.75	4.75-< 5.5	5.5-< 6.25	6.25-< 7
1-< 1.75	0	0.335	0.366	0.442	0.73	1	1	1
1.75-< 2.5	0.335	0	0.032	0.107	0.395	0.805	1	1
2.5-< 3.25	0.366	0.032	0	0.075	0.364	0.774	1	1
3.25-< 4	0.442	0.107	0.075	0	0.288	0.698	1	1
4-< 4.75	0.73	0.395	0.364	0.288	0	0.41	0.738	0.918
4.75-< 5.5	1	0.805	0.774	0.698	0.41	0	0.328	0.508
5.5-< 6.25	1	1	1	1	0.738	0.328	0	0.18
6.25-< 7	1	1	1	1	0.918	0.508	0.18	0

Segmentation Building Summary

The Segmentation Building Summary table shows the number of iterations in the segmentation process and the segmentation status after each iteration. The Estimated total score is a measure of the overall segmentation strength.

The Segmentation Building Summary table is only shown if the `DETAILS` option is specified.

Segmentation Building Summary			
Iteration	Segments	Movements	Estimated total score
1	6	6	.
2	3	147	0.6981
3	4	5	0.6987
4	3	6	0.7025
5	3	0	0.7034

Segmentation Summary

The `Segmentation Summary` table shows the final distribution of observations in each segment. The `Strength` value is a measure of the segmentation strength in each segment.

Segmentation Summary		
Segment	Members	Strength
1	63	0.643
2	54	0.709
3	33	0.726

Relative Entropy by Segment and Variable

The `Relative Entropy by Segment and Variable` table shows the relative entropy for each variable in each segment. The values shown are measures of the difference between the distribution of values for that variable in the segment and in the dataset as a whole. The larger the value for a variable, the greater the difference between the values of that variable in the segment and the values of that variable in the distribution as a whole. The calculated value is the *Kullback–Leibler divergence* described by Kullback (Kullback, 1959 [3]).

Relative Entropy by Segment and Variable					
Segment	sepal_length	sepal_width	petal_length	petal_width	species
1	0.587	0.428	0.95	0.92	0.744
2	1.138	0.562	1.365	1.359	1.204
3	1.427	0.266	1.722	1.682	1.585

Segment Plots

There is a segment plot for each segment. Each plot shows the proportion of observations in that segment, followed by a bar chart for each variable in the segmentation. Each bar chart shows the distribution of values of that variable in that segment as a coloured bar and the distribution of values of that variable in the population as a whole as a grey bar.

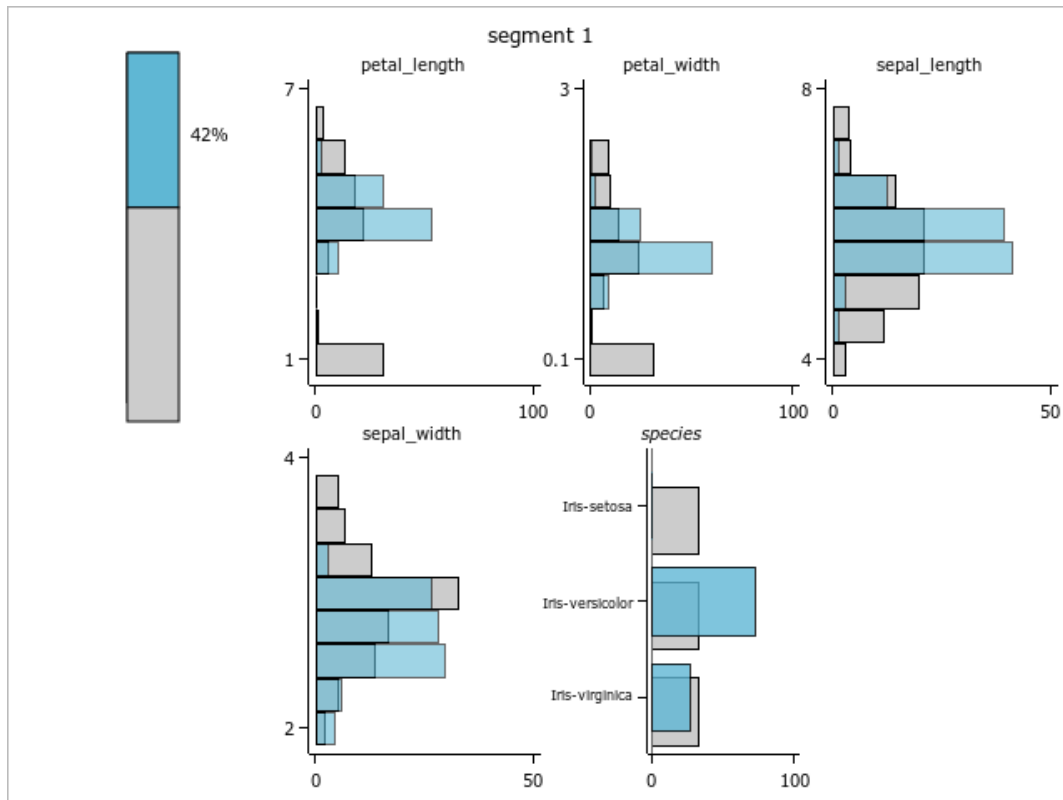
The bar charts are shown in order of significance of the variable in the segment. The most significant variable in a segment is the one where the distribution of the values of the variable in the segment is most dissimilar to the distribution of the values of the variable in the overall population.

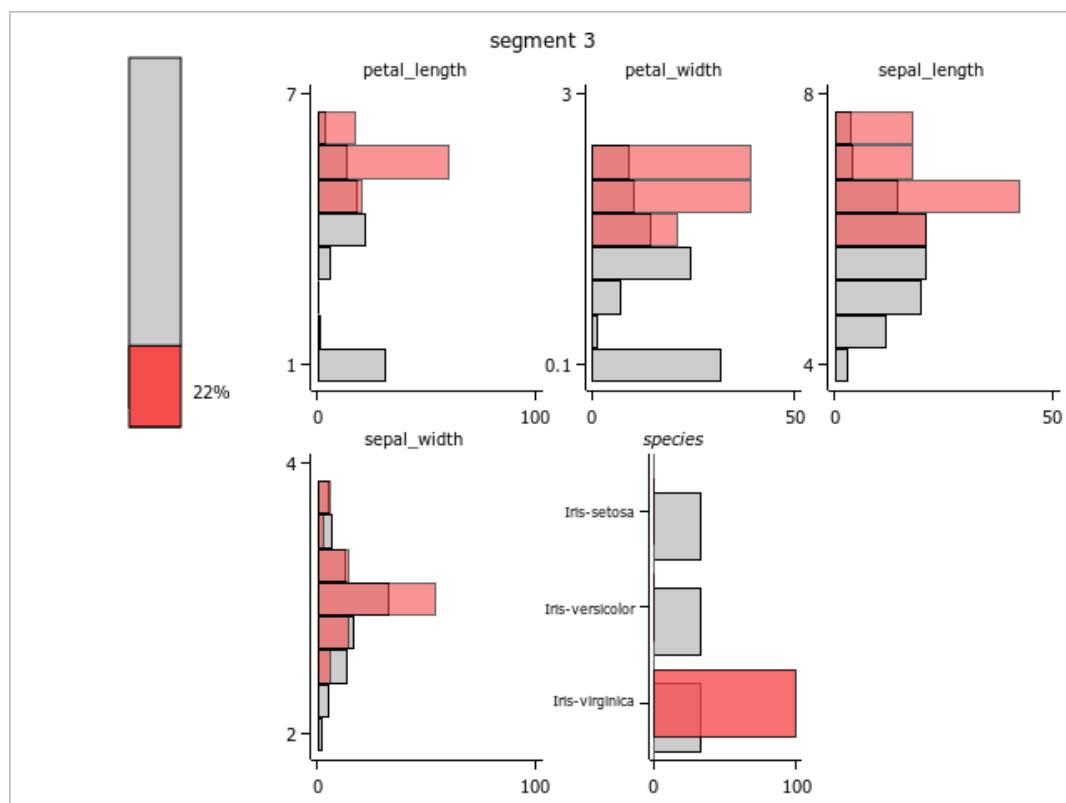
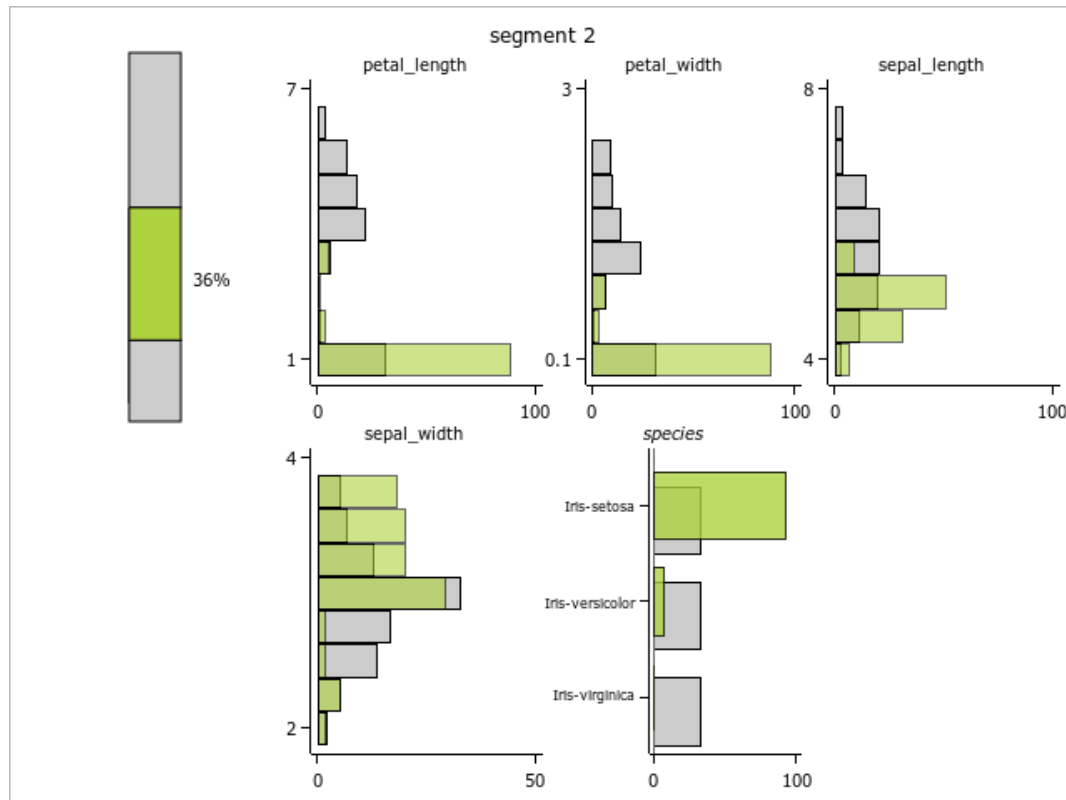
In this example, the first segment contains 42% of the overall observations, the second segment contains 36% of the observations and the third segment contains 22%.

The most significant variable in the first segment is `petal_length` (the `Relative Entropy by Segment and Variable` table above shows that in segment 1, `petal_length` has a relative entropy of 0.95, which is the highest value for that segment). Looking at the fifth bar of this histogram (corresponding to bin $4 < 4.75$, according to the `Differences for petal_length` table) you can see that around 50% of the observations in this segment have a petal length in that range, whereas only around 20% of observations in the overall population have that value. And, looking at the first bar of this histogram, (bin $1 < 1.75$) you can see that no observations in this segment have a petal length in that range, but around 30% of observations in the overall population have that value.

In this example, you can see how the segmentation has split the observations into similar groups. For example, the first segment has more mid-range values for `petal_length`, `petal_width` and `sepal_length` than in the overall population, and more high values for `sepal_width` than in the overall population. The second segment has more high values for `petal_length`, `petal_width` and `sepal_length` than in the overall population, and more low values for `sepal_width` than in the overall population. The third segment has more low values for `petal_length`, `petal_width` and `sepal_length` than in the overall population, and more mid-range values for `sepal_width` than in the overall population.

The final bar chart in each segment shows the distribution of values of the `species` variable in the segment. This variable was specified as `REPORTONLY`, so was not considered in the segmentation calculations. Here, you can see that, in the first segment, there are no `Iris-setosa` observations, but around 75% of the observations are `Iris-versicolor` and around 25% of the observations are `Iris-virginica`. In the second segment, almost all the observations are `Iris-setosa`, and in the third segment, all the observations are `Iris-virginica`.





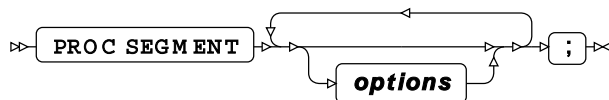
SEGMENT procedure reference

Describes the syntax and options for `PROC SEGMENT` and its contained statements.

<code>PROC SEGMENT</code> ↗	3329
Determines the optimal way to segment a dataset and reports the similarities and differences between the observations in each segment.	
<code>BY</code> ↗	3339
Groups the observations in the input dataset using one or more specified variables.	
<code>FREQ</code> ↗	3339
Specifies a variable containing the frequency associated with an observation.	
<code>INPUT</code> ↗	3339
Specifies the variables to be included in the segmentation and the options that apply to the variables.	
<code>OUTPUT</code> ↗	3345
Outputs a dataset containing the data that was processed and the segmentation that was applied.	
<code>SCORE</code> ↗	3346
Uses the current segmentation to score the data in the specified dataset.	
<code>SELECT</code> ↗	3348
Specifies a SAS language expression to select the observations from the dataset to be segmented.	
<code>WHERE</code> ↗	3348
Restricts the observations to be processed.	

PROC SEGMENT

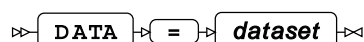
Determines the optimal way to segment a dataset and reports the similarities and differences between the observations in each segment.



SEGMENT options

The following *options* are available:

DATA



Specifies the dataset to be segmented.

The `DATA` option cannot be specified if the `INMODEL` option is also specified. If neither the `DATA` nor the `INMODEL` options are specified, the most recently-created dataset is used as the training dataset.

If `PROC SEGMENT` includes a `BY` statement, the dataset to be segmented must contain all the variables specified in the `BY` statement, and must be sorted in order of those variables.

DETAILS

DETAILS

Specifies that the ODS output includes the `Segmentation Building Summary` table, which shows the segment modifications and the segmentation strength after each iteration of the segmentation.

By default, the `Segmentation Building Summary` table is not included in the ODS output.

DIFFERENCES

DIFFERENCES

Specifies that the ODS output includes the `Differences` tables, which show, for each input variable, the calculated difference between each pair of values of a discrete variable or the bins of a continuous variable. These are the values that are used in the segmentation process to maximise the similarities between observations in a segment and maximise the differences between observations in different segments. The table is not printed for variables with large numbers of values (those exceeding `MAXSUMMARYBARS`).

By default, the `Differences` tables are not included in the ODS output.

DISTRIBUTIONS

DISTRIBUTIONS

Specifies that the ODS output includes the `Profile by Segment` tables, which show, for each input variable, the distribution of values across each segment.

By default, the `Profile by Segment` tables are not included in the ODS output.

ENTROPY

ENTROPY

Specifies that the ODS output includes the `Relative Entropy by Segment and Variable` table. This table shows the relative entropy of each variable in each segment. The relative entropy of a variable is a measure of the difference between the distribution of values for that variable in the segment and in the dataset as a whole. The calculated value is the *Kullback–Leibler divergence* described by Kullback (Kullback, 1959 [3]).

By default, the `Relative Entropy by Segment and Variable` table is not included in the ODS output.

FAST

▷▷ **FAST** ◁◁

Specifies that the segmentation process is optimised for speed. For a large dataset, this option improves performance, but may produce less accurate results. For example, one of the optimisations is that, for a continuous variable, the calculations assume that all the observations in a bin have the same value as the value of the variable at the centre of the range of the bin.

By default, the segmentation process is optimised for accuracy not speed.

INEST

▷▷ **INEST** = *dataset* ◁◁

Specifies a dataset saved by the `OUTEST` option in a previous call of `PROC SEGMENT` containing a saved segmentation. The segmentation specified in this dataset can be used as the initial definition for segmentation or scoring.

The dataset saved by the `OUTEST` option does not include the segmentation parameters, so you should use the `INPUT` statement to specify these. It is recommended that you use the same options that were used when the saved segmentation was created, although you can use different options.

If the `DATA` option is also specified, `PROC SEGMENT` uses the data in the dataset specified by the `DATA` option to refine the segmentation. If both the `DATA` option and the `NOFIT` option are specified the observations in the dataset specified by the `DATA` option are ignored in the segmentation calculations, but can be scored or output.

INMODEL

▷▷ **INMODEL** = *dataset* ◁◁

Specifies a dataset saved by the `OUTMODEL` option in a previous call of `PROC SEGMENT`. The segmentation and controlling parameters saved in this dataset can be used to score another dataset.

If any `INPUT` statements are specified they are ignored, since all required parameters are defined in the `INMODEL` dataset.

The `INMODEL` option cannot be specified if `DATA` is also specified.

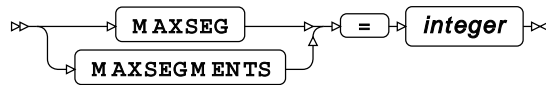
MAXITER

▷▷ **MAXITER** = *integer* ◁◁

Specifies the maximum number of iterations to use during the segmentation calculations. The specified value must be a positive integer. The default value is 25.

This option is ignored if `INMODEL` is also specified.

MAXSEG



Specifies the maximum number of segments that can be created in an iteration during the segmentation calculations. If the specified number of segments is reached, no further segments are created and any remaining observations are assigned to the closest existing segment.

The default value is 100.

This option is ignored if `INMODEL` is also specified.

MAXSUMMARYBARS

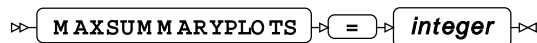


Specifies the maximum number of bars to show for a variable in the summary plot histograms for each segment.

If a discrete variable has more distinct values than the specified `MAXSUMMARYBARS`, the data for the least significant bars are combined. For each segment, the bars that are combined are the bars corresponding to the values where the distribution of the variable values in the segment and the distribution of the variable values in the overall population are the most similar. The combined bars are labelled with the label specified in the `OTHERLABEL` option, or are labelled `Other`, if the `OTHERLABEL` option is not specified.

The default value is `MAXSUMMARYBARS` is 20.

MAXSUMMARYPLOTS

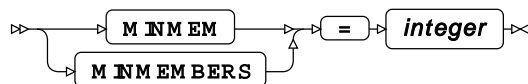


Specifies the maximum number of graphs to show for each segment in the ODS output.

If the number of variables in the segmentation is more than `MAXSUMMARYPLOTS`, the graphs for the least significant variables in each segment are omitted. For each segment, the least significant variables are those where the distribution of the variable values in the segment and the distribution of the variable values in the overall population are the most similar.

The default value of `MAXSUMMARYPLOTS` is 12.

MINMEM



Specifies the minimum number of members in a segment in order for the segment to be shown in the ODS output. Segments with fewer than the specified number of members are discarded.

The default value is 1.

This option is ignored if `INMODEL` is also specified.

MINPERCENT

➤ `MINPERCENT` ➤ `=` ➤ *integer* ➤

Specifies the minimum number of members in a segment in order for the segment to be shown in the ODS output, as a percentage of the observations in the dataset. Segments with fewer than the specified percentage of members are discarded.

The default value is 5 (percent).

This option is ignored if `INMODEL` is also specified.

NOFIT

➤ `NOFIT` ➤

Specifies that the segmentation of the data is not created from the dataset specified in the `DATA` option. You can use this option in conjunction with the `INEST` option to specify that the saved segmentation is used instead. The dataset specified in the `DATA` option is not used to create a segmentation, but can be scored or used to create an output dataset.

By default, the initial segmentation of the data is created from the data in the input dataset.

`NOFIT` cannot be specified with `OUTMODEL`. If `NOFIT` is specified with `INMODEL`, `NOFIT` is ignored.

NOPRINT

➤ `NOPRINT` ➤

Suppresses all printed output from `PROC SEGMENT`. By default the printed output includes information about each of the variables and other options specified for the segmentation, information about the segmentation, and a number of plots showing the way that observations have been allocated to segments.

`NOPRINT` overrides the other print options (`DETAILS`, `DIFFERENCES`, `DISTRIBUTIONS`, `ENTROPY`, `PROFILES`, `SIMILARITIES`) and the `PLOT` option.

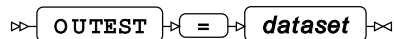
OTHERLABEL

➤ `OTHERLABEL` ➤ `=` ➤ *label* ➤

Specifies the label to use for the 'other' category for discrete variables in a segmentation plot . This value is overridden if a variable-specific label is specified in the `OTHERLABEL` option of the `INPUT` statement.

The default label for the 'other' category is `Other`.

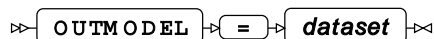
OUTEST



Specifies that the segmentation definition is saved in a dataset. The saved information includes the distribution of the data used to create the segmentation, and the segmentation of that data. The definition can later be used as the initial definition to segment another dataset.

If the `NOFIT` option is also specified, the saved information just includes the distribution of the data but not the segmentation.

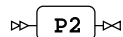
OUTMODEL



Specifies that the segmentation model is saved in a dataset. The saved information includes the segmentation parameters, the distribution of the data used to create the segmentation, and the segmentation of that data. The information in this dataset can later be used to score another dataset.

The `NOFIT` option cannot be specified with `OUTMODEL`.

P2



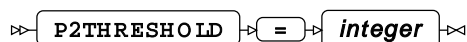
Specifies that a variant of the P^2 algorithm (Jain and Chlamtac, 1985 [2]) is used to estimate the order statistics for variable values. This option allows you to override the default behaviour if the number of observations is likely to be large but is not known at the outset.

If `P2` is not explicitly specified, the method used to derive the order statistics for the variable values depends on the number of observations in the dataset being segmented:

- If the number of observations is not known at the outset (for example, if the observations are being selected from a database) or if the number of observations is known and is less than `P2THRESHOLD`, the order statistics are derived by storing and ordering all the observations.
- If the number of observations is known and greater than `P2THRESHOLD`, the order statistics are estimated using the P^2 algorithm.

This option is ignored if `INMODEL` is also specified.

P2THRESHOLD



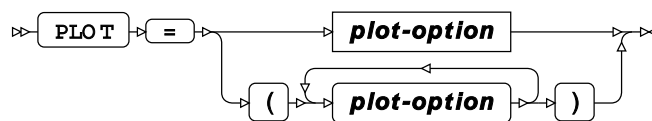
Specifies the threshold number of observations above which a variant of the P^2 algorithm (Jain and Chlamtac, 1985 [2]) is used to estimate the order statistics for variable values. This avoids memory problems when processing very large datasets. If the number of observations in the dataset is less than the specified threshold value, the order statistics are derived by storing and ordering all the observations.

The default value is 100,000.

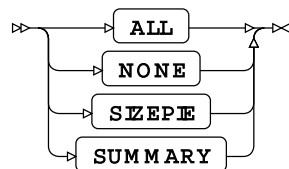
This option is ignored if `P2` is also specified.

This option is ignored if `INMODEL` or `INEST` is also specified.

PLOT



plot-option



Specifies the segmentation plots that are produced through the ODS graphics framework.

If the segmentation is derived from an input dataset, the default is `PLOT=SUMMARY`. If a previously generated segmentation is loaded using the `INMODEL` option, the default is `PLOT=NONE`.

ALL

All segmentation plots are produced.

`ALL` overrides `SIZEPIE` and `SUMMARY` if also specified.

NONE

No segmentation plots are produced.

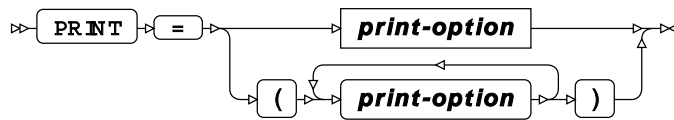
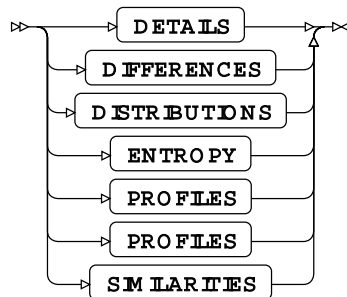
`NONE` overrides `ALL`, `SIZEPIE` and `SUMMARY` if also specified.

SIZEPIE

A single pie chart is produced showing the proportion of observations in each segment.

SUMMARY

A series of plots are produced for each segment showing the distribution of the values of each variable in the segment and, for comparison, the distribution of the values of each variable in the whole population.

PRINT**print-option**

Specifies the optional segmentation tables to include in the ODS output.

By default, none of the optional segmentation tables are included.

DETAILS

Specifies that the ODS output includes the `Segmentation Building Summary` table, which shows the segment modifications and the segmentation strength after each iteration of the segmentation.

`PRINT=DETAILS` is an alias of `DETAILS`.

DIFFERENCES

Specifies that the ODS output includes the `Differences` tables, which show, for each input variable, the calculated difference between each pair of values of a discrete variable or the bins of a continuous variable. These are the values that are used in the segmentation process to maximise the similarities between observations in a segment and maximise the differences between observations in different segments. The table is not printed for variables with large numbers of values (those exceeding `MAXSUMMARYBARS`).

`PRINT=DIFFERENCES` is an alias of `DIFFERENCES`.

DISTRIBUTIONS

Specifies that the ODS output includes the `Profile by Segment` tables, which show, for each input variable, the distribution of values across each segment.

`PRINT=DISTRIBUTIONS` is an alias of `DISTRIBUTIONS`.

ENTROPY

Specifies that the ODS output includes the `Relative Entropy by Segment and Variable` table. This table shows the relative entropy of each variable in each segment. The relative entropy of a variable is a measure of the difference between the distribution of values for that variable in the segment and in the dataset as a whole. The calculated value is the *Kullback–Leibler divergence* described by Kullback (Kullback, 1959 [3]).

PRINT=ENTROPY is an alias of ENTROPY.

PROFILES

Specifies that the ODS output includes the `Variable Profile` tables, which show, for each input variable, the percentage frequency of the values of the variable in the data. The frequencies are derived from the input dataset, or from the `INEST` or `INMODEL` datasets, as applicable.

PRINT=PROFILES is an alias of PROFILES.

SIMILARITIES

Specifies that the ODS output includes the `Approximate Segment Similarity` table. This shows the overall similarity value between the observations in each pair of segments. The leading diagonal of this table contains the similarity values for observations in the same segment. These values are the same as the segmentation strength values in the `Segmentation Summary` table. The off-diagonal entries are the similarity values between observations in different segments. The overall score is derived from these similarity values and is the value that the segmentation attempts to optimise.

PRINT=SIMILARITIES is an alias of SIMILARITIES.

PROFILES

➤ PROFILES ➤

Specifies that the ODS output includes the `Variable Profile` tables, which show, for each input variable, the percentage frequency of the values of the variable in the data. The frequencies are derived from the input dataset, or from the `INEST` or `INMODEL` datasets, as applicable.

By default, the `Variable Profile` tables are not included in the ODS output.

SEGMENTFORMAT

➤ SEGMENTFORMAT ➤ = ➤ *format* ➤

Specifies a format to be associated with the segment number in the output data and in ODS output. This can be used in conjunction with `PROC FORMAT` to give meaningful labels to the segments.

The specified format must be a numeric format.

By default, the segment number is displayed using the default numeric format.

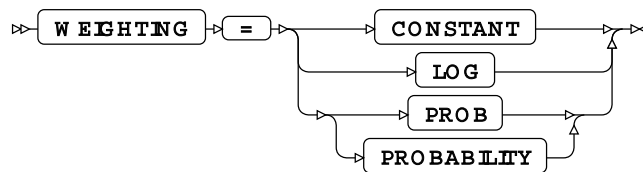
SIMILARITIES

➤ SIMILARITIES ➤

Specifies that the ODS output includes the `Approximate Segment Similarity` table. This shows the overall similarity value between the observations in each pair of segments. The leading diagonal of this table contains the similarity values for observations in the same segment. These values are the same as the segmentation strength values in the `Segmentation Summary` table. The off-diagonal entries are the similarity values between observations in different segments. The overall score is derived from these similarity values and is the value that the segmentation attempts to optimise.

By default, the `Approximate Segment Similarity` table is not included in the ODS output.

WEIGHTING



Specifies the general weighting function to apply to pairs of observations when a variable has the same value in both observations. The weighting determines the degree of importance to attach to similarities and differences between common and rarer values of a variable when assigning observations to segments.

The default weighting is `CONSTANT`.

This option is ignored if `INMODEL` is also specified.

CONSTANT

All agreements between variable values in pairs of observations contribute an equal weight to the segmentation calculation. This means that overall, common pairings contribute more weight to the segmentation than uncommon pairings, because there are more of them. For more information, see *CONSTANT weighting* [↗](#) (page 3321).

LOG

The values in all pairs of observations contribute a weight to the segmentation that is proportional to the negative logarithm of the probability of that pairing occurring. This means that the effect of pairings of values of a variable is proportional to the amount of information in that pairing. For more information, see *LOG weighting* [↗](#) (page 3321).

Note:

This option is experimental.

PROB

The values of the variables in all pairs of observations contribute a weight to the segmentation that is inversely proportional to the probability of that pairing occurring. This means that rare pairings of values of a variable have relatively more effect on the segmentation than common pairings, so overall, every pairing of variables contributes an

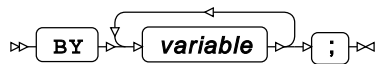
equal amount. Each instance of an uncommon pair of values contributes more weight, but there are fewer of them. Similarly, each instance of a common pair of values contributes less weight, but there are more of them. For more information, see *PROBABILITY weighting* [↗](#) (page 3321).

Note:

This option is experimental.

BY

Groups the observations in the input dataset using one or more specified variables.

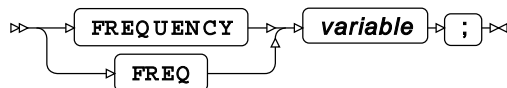


The specified variable or variables are used to separate the input data into groups. PROC `SEGMENT` generates a separate segmentation model from the data in each group.

If the `BY` statement is included, the input dataset must be pre-sorted on the specified variable or variables.

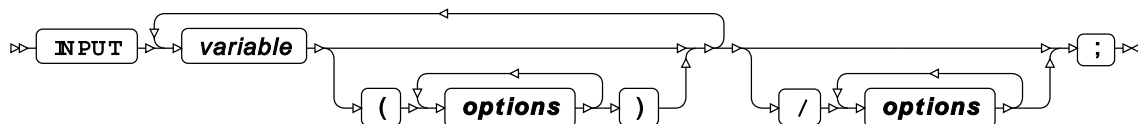
FREQ

Specifies a variable containing the frequency associated with an observation.



INPUT

Specifies the variables to be included in the segmentation and the options that apply to the variables.



Options can be specific to a single variable or can be global. Options in brackets after a variable name are specific to that variable. Options specified after the forward slash are global options and apply to all input variables in the `INPUT` statement. Variable-specific options override global options. Unless otherwise specified, all options can be variable-specific or global.

Some options are applicable to any input variable; others are only applicable to discrete input variables and still others are only applicable to continuous input variables. By default, all numeric variables are regarded as continuous and all character variables are regarded as discrete unless the `COMPARE` option specifies something different.

More than one `INPUT` statement can be present.

If no `INPUT` statement is present, all variables in the dataset are included in the segmentation.

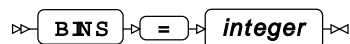
variable

Specifies a variable to include in the segmentation.

Options

The following *options* are available:

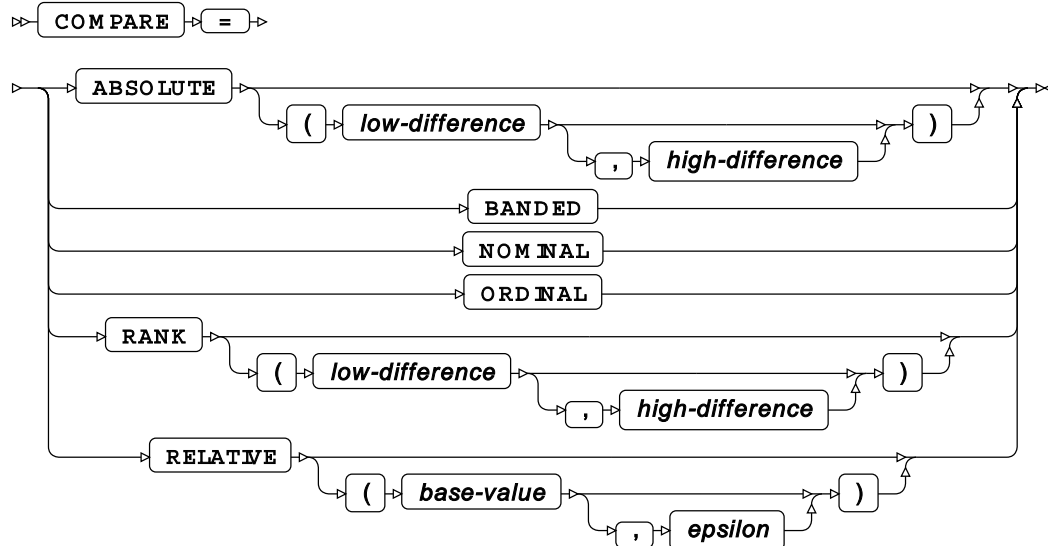
BINS



For a continuous variable, specifies the number of bins into which to divide the range. A larger number of bins can give a more accurate segmentation.

The default value is 10 and the minimum value is 2. This option is ignored for discrete variables.

COMPARE



Specifies the kind of variable and the method used to compare values of the variable when determining the segmentation.

`ABSOLUTE`, `RANK` and `RELATIVE` specify that the variable is treated as continuous and `BANDED`, `NOMINAL` and `ORDINAL` specify that the variable is treated as discrete.

For numeric variables, the default value is `RANK (0, 0.5)` and for character variables, the default value is `NOMINAL`.

ABSOLUTE

The variable is a continuous variable.

The difference between two values is a function of the absolute difference between the two values. *low-difference* specifies the absolute difference below which two values are considered identical, with a difference value of 0 (zero). *high-difference* specifies the absolute difference above which two values are considered completely different, with a difference value of 1. Pairs of values with absolute differences between *low-difference* and *high-difference* have difference values calculated linearly between 0 and 1.

For more information, see *ABSOLUTE comparisons* [↗](#) (page 3317).

If only *low-difference* is specified, then *high-difference* is the same as *low-difference*. If the absolute difference between two values is less than *low-difference*, the two values are considered identical, with a difference value of 0 (zero). If the absolute difference between two values is greater than *low-difference*, the two values are considered completely different, with a difference value of 1. If the absolute difference between two values is equal to *low-difference*, the difference value is 0.5.

If neither *low-difference* or *high-difference* are specified, then *low-difference* and *high-difference* both default to the standard deviation of the variable. The standard deviation is calculated from the original values of the variable, before any adjustments specified by the `MIN`, `MAX` or `WINSOR` options have been made to extreme values.

BANDED

The variable is discrete and grouped into a number of ordered bands.

The difference between two values is measured by the estimated proportion of the population between the two values. Two values in the same band have a difference value of slightly greater than 0. Two values, where one is in the lowest band and the other is in the highest band, have a difference value of slightly less than 1. This reflects that fact that the underlying values are likely to be slightly different, although they are in the same band.

For more information, see *BANDED comparisons* [↗](#) (page 3317).

NOMINAL

The variable is a discrete variable with no explicit ordering.

Two values are considered identical, with a difference value of 0 (zero), if they are identical. Otherwise they are considered completely different, with a difference value of 1.

For more information, see *NOMINAL comparisons* [↗](#) (page 3318).

The external (formatted) values are compared unless `ORDER=INTERNAL` is also specified.

ORDINAL

The variable is a discrete variable with an explicit ordering.

Two values are considered identical, with a difference value of 0 (zero), if they are identical. Two values are considered completely different, with a difference value of 1, if one is in the lowest category and one is in the highest category. Otherwise the difference is measured by the fraction of the number of categories between them.

For more information, see *ORDINAL comparisons* [↗](#) (page 3318).

The internal (unformatted) values are compared unless `ORDER=EXTERNAL` is also specified.

RANK

The variable is a continuous variable.

The difference between two values is measured by the proportion of the population between the two values. *low-difference* specifies the proportion of the population below which two values are considered identical, with a difference value of 0 (zero). *high-difference* specifies the proportion of the population above which two values are considered completely different, with a difference value of 1. Pairs of values with differences between *low-difference* and *high-difference* have difference values calculated linearly between 0 and 1.

For more information, see *RANK comparisons* [↗](#) (page 3318).

If neither *low-difference* or *high-difference* are specified, then *low-difference* defaults to 0, and *high-difference* defaults to 0.5.

If only *low-difference* is specified, then *high-difference* is the same as *low-difference*. If the proportion of the population between two values is less than *low-difference*, the two values are considered identical, with a difference value of 0 (zero). If the proportion of the population between two values is greater than *low-difference*, the two values are considered completely different, with a difference value of 1. If the proportion of the population between two values is equal to *low-difference*, the difference value is 0.5.

RELATIVE

The variable is a continuous variable.

The difference between two values is determined by comparing their absolute difference with a value based on the root mean square of the two values. If the absolute difference is less than the root mean square value, the two values are considered identical, with a difference value of 0 (zero). If the absolute difference is equal to the root mean square value, the difference value is 0.5. Otherwise, the two values are considered completely different with a difference value of 1. *base-value* specifies an offset to apply to each value before calculating the root mean square, and *epsilon* specifies a scale factor to apply to the calculated root mean square value before comparing it with the absolute value.

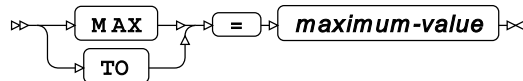
For more information, see *RELATIVE comparisons* [↗](#) (page 3319).

The default *epsilon* is 1 and the default *base-value* is 0.

DESCENDING

For a discrete variable, specifies that the sort order is reversed.

This option is ignored for continuous variables.

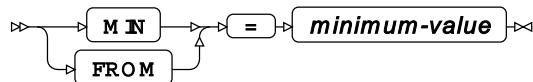
MAX

For a continuous variable, specifies the highest value of the variable to retain for the segmentation calculations. Observations with values higher than the specified maximum value have the value of the variable replaced with the specified maximum value.

If **MAX** is specified, the specified value overrides the upper limit derived from winsorization.

If not specified, the default value is the value determined by the winsorization parameter specified in the **WINSOR** option, or if the **WINSOR** option is not specified, the value determined by the default winsorization parameter value of 0.9.

This option is ignored for discrete variables.

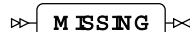
MIN

For a continuous variable, specifies the lowest value of the variable to retain for the segmentation calculations. Observations with values lower than the specified minimum value have the value of the variable replaced with the specified minimum value.

If **MIN** is specified, the specified value overrides the lower limit derived from winsorization.

If not specified, the default value is the value determined by the winsorization parameter specified in the **WINSOR** option, or if the **WINSOR** option is not specified, the value determined by the default winsorization parameter value of 0.9.

This option is ignored for discrete variables.

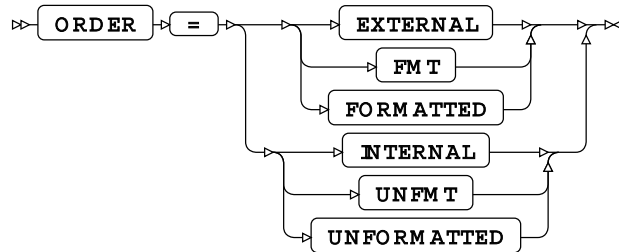
MISSING

For a discrete variable, specifies that missing values are treated as valid values.

If not specified, missing values are ignored in the segmentation calculations. Other variables in the observation with non-missing values are used as normal in the segmentation calculations.

This option is ignored for continuous variables. Any observation with a missing value for a continuous variable is always ignored in the segmentation calculation for that variable.

ORDER



For a discrete variable, specifies the sort order to use.

The default value is `EXTERNAL`. This option is ignored for continuous variables.

EXTERNAL

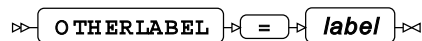
The variable is ordered by the external (formatted) values using alphanumeric ordering. If the `DESCENDING` option is also specified, the sort order is descending, otherwise the sort order is ascending.

INTERNAL

The variable is ordered by the internal (unformatted) values using alphanumeric ordering.

If the `DESCENDING` option is also specified, the sort order is descending, otherwise the sort order is ascending.

OTHERLABEL



For a discrete variable, specifies the label to use for the 'other' category in the segmentation plots for that variable. This value overrides any general label specified in the `OTHERLABEL` option of `PROC SEGMENT`.

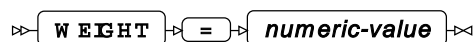
This option is ignored for continuous variables.

REPORTONLY



Specifies that the variable is omitted from the segmentation calculations but is included in the distribution profiles and segmentation plots.

WEIGHT



Specifies the variable-specific weighting value to apply to the variable. It determines the degree of importance to attach to similarities and differences in the values of the variable, relative to similarities and differences in the values of the other variables.

WINSOR

➤ **WINSOR** ➤ = ➤ *numeric-value* ➤

For a continuous variable, specifies the value used to *winsorize* (remove extreme values of) the variable in the segmentation calculations. The specified value is the (two-tailed) proportion of the ordered values of the variable to retain, assuming a normal distribution.

The **MAX** option, if specified, overrides the upper limit derived from winsorization. The **MIN** option, if specified, overrides the lower limit derived from winsorization.

The default winsorization value is 0.9, which replaces the lowest 5% of values with the estimated value at the 5th percentile and highest 5% of values with the estimated value at the 95th percentile.

This option is ignored for discrete variables.

OUTPUT

Outputs a dataset containing the data that was processed and the segmentation that was applied.

➤ **OUTPUT** ➤ ➤ *options* ➤ ➤ ; ➤

Options

The following *options* are available:

OUT

➤ **OUT** ➤ = ➤ *dataset* ➤

Specifies the name of the output dataset.

If **OUT** is not specified, the procedure creates the dataset as *DATAN* in the **WORK** library, where *n* is incremented for each output dataset.

SCORE

➤ **SCORE** ➤ = ➤ *varname* ➤

Specifies the name of a variable in the output dataset to contain the score value for the observation. The score value indicates how similar the observation is to the other observations in that segment.

By default, the score value is not included in the output dataset.

SEGMENT

➤ **SEGMENT** ➤ = ➤ *varname* ➤

Specifies the name of a variable in the output dataset to contain the segment number the observation is most likely to belong to.

The default variable name is `__SEGMENT__`.

SCORE

Uses the current segmentation to score the data in the specified dataset.

➤ **SCORE** ➤ *options* ➤ ; ➤

The **SCORE** statement takes the data in the specified dataset and scores it using the segmentation defined in **PROC SEGMENT**. The score value indicates how similar the observation is to the other observations in that segment.

If no dataset is specified in the **SCORE** statement, the dataset in the **PROC SEGMENT** statement is scored.

By default, the scored dataset contains the input observations, and for each, the segment the observation is most likely to belong to. All observations are scored, even those with missing values.

Multiple **SCORE** statements can be specified.

Options

The following *options* are available:

COUNT

➤ **COUNT** ➤ = ➤ *integer* ➤

Specifies the number of potential segments to be included in the scored dataset for each observation. The segment number with the highest score is output in the variable specified by the **SEGMENT** option. The segment with the next highest score is output in the specified variable, with 2 appended to the name. Subsequent segment numbers are output in order of decreasing scores, in variables with increasing suffixes.

If the **SCORE** option is also specified then the score value is also output for each potential segment.

The default number of potential segments included in the scored dataset is 1.

DATA

DATA = dataset

Specifies the dataset to score. The dataset to score must contain all variables specified in the `INPUT` statement, or, if the `PROC SEGMENT INMODEL` option is used, the dataset to score must contain all variables saved in the `INMODEL` dataset.

If `PROC SEGMENT` includes a `BY` statement, the dataset to score must also contain all the variables mentioned in the `BY` statement, and must be sorted in the order of those variables.

If the `DATA` option is not present, the dataset in the `PROC SEGMENT` statement is used.

OUT

OUT = dataset

Specifies the dataset to which the score results are output. The dataset contains the original data, and additional fields showing which segment each observation is most likely to belong to.

If `OUT` is not specified, the dataset name is `WORK.DATAN`, where *n* is incremented for each output dataset.

If `PROC SEGMENT` specifies more than one output dataset for score results (for example, if there is more than one `SCORE` statement) then each output dataset must have a unique name.

SCORE

SCORE = varname

Specifies the name of a variable in the scored dataset to contain the score value for an observation. The score value indicates how similar the observation is to the other observations in that segment.

By default, the score value is not included in the scored dataset.

SEGMENT

SEGMENT = varname

Specifies the name of a variable in the scored dataset to contain the segment number the observation is most likely to belong to.

The default variable name is `__SEGMENT__`.

SELECT

Specifies a SAS language expression to select the observations from the dataset to be segmented.



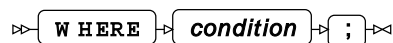
ⁱ See *SAS Language expressions* [↗](#) (page 24).

The `SELECT` statement uses the entire dataset to derive the overall distribution of observations. It then selects the specified observations, calculates the segment allocations and displays the segment distribution for those observations against the overall distribution of all observations. This is different from the `WHERE` statement, which derives both the overall distribution of observations and the segment allocation from the specified observations.

The `SELECT` statement can only appear once in `PROC SEGMENT`.

WHERE

Restricts the observations to be processed.



The `WHERE` statement derives both the overall distribution of observations and the segment allocation from the specified observations. This is different from the `SELECT` statement, which uses the entire dataset to derive the overall distribution of observations, then selects the specified observations, calculates the segment allocations and displays the segment distribution for those observations against the overall distribution of all observations.

SEGMENT bibliography

These items are referenced in the `SEGMENT` procedure section.

- [1] Michaud, P., 1995. Variational data analysis versus classical data analysis. In: J.Jansen et al., eds. *Advances in Stochastic Modelling and Data Analysis*, pp. 128–158.
- [2] Jain, R. & Chlamtac, I., 1985. The P2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Storing Observations. In: R. Sargent, ed. *Communications of the ACM*, Volume 28, pp. 1076–1085.
- [3] Kullback, S., 1959, *Information Theory and Statistics*, John Wiley and Sons.
- [4] Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), pp. 179–188..

SVM procedure

The SVM procedure enables you to build a support vector machine (SVM) from an input dataset. You can then use the SVM model to analyse other datasets.

About support vector machines

A support vector machine (SVM) is a supervised learning model that is capable of analysing data for classification or regression.

The SVM procedure supports classification and regression models. SVM modelling relates a response variable, y , to one or more predictor variables, x , where y can be a class (for classification models) or a value (for regression models).

Specifically, the one class SVM model, binary classification model and multi-class classification versions of the model are supported for classification purposes.

Development of Support Vector Machines was largely pioneered at AT&T Bell Laboratories by Boser, et al., 1992 [1], Guyon, et al., 1993 [2], Cortes & Vapnik, 1995 [3], Scholkopf, et al., 1995 [4], Scholkopf, et al., 1996 [5] and Vapnik, et al., 1997 [6].

SVM models were first developed by Vapnik & Lerner, 1963 [7] such that they involved the construction of a linear classifier. However, Boser, et al., 1992 [1] suggested a way to extend this to non-linear classification via the application of the kernel trick (Aizerman, et al., 1964 [8]). This enables the algorithm to construct a linear classifier in a transformed feature space that may be non-linear in the original input predictor space. The SVM procedure supports the most commonly used kernel types for both classification and regression modelling. For more information see the `KERNEL` option of the `MODEL` statement.

SVM classification

For classification models, binary, multi-class and single class models are supported.

Binary classification models

A binary classification model maps predictor variables to a response variable in one of two classes.

In a binary classification model, the response, y , will be one of two elements.

A linear binary SVM model attempts to find a hyperplane that optimally partitions the predictor space into two classes. The assumption is that the predictor space can be partitioned using a hyperplane (the training data is linearly separable). This gives us the following score function:

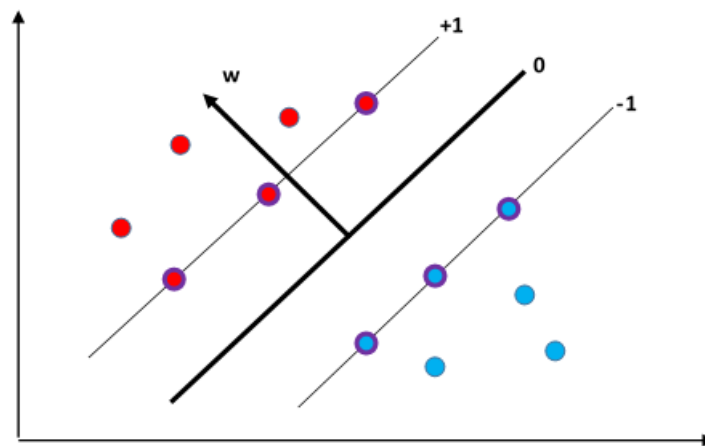
$$s(\vec{x}) = \vec{x}^T \vec{w} + b$$

where:

- x is an observation in predictor space
- w is the orthogonal vector to the hyperplane that separates the predictor space in two
- b is a bias term.

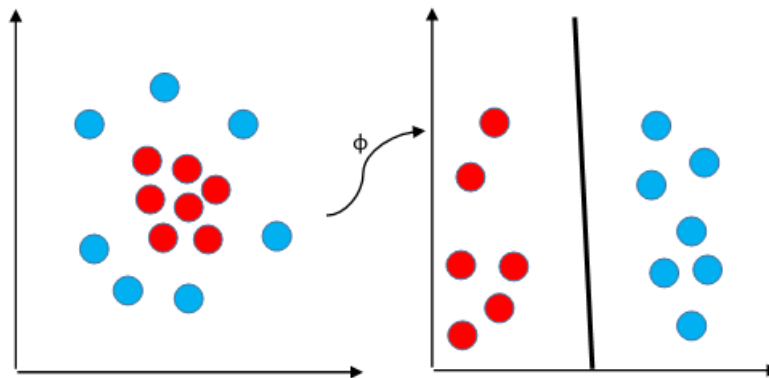
The score function s provides a function for assigning predictor variables to one of the two classes; a negative value mapping the predictors to one class, and a non-negative value mapping the predictors to the other class.

Figure 376. Two classes of data, linearly-separated by a binary classifier



The SVM procedure attempts to find a w and a b that can be used to correctly classify as much of the predictor space as possible. It does this by using a provided set of training observations. Each observation in the training dataset includes its position in the predictor space, x , as well as the class to which it belongs, y .

Thus far it has been assumed that the training data is linearly separable in the predictor space. For many collections of data this assumption is incorrect and, for the SVM model to work in these cases, the predictor space has to be transformed so that the data becomes linearly separable. This transformation occurs via a *kernel function*. A kernel function represents how one predictor variable projects onto another in the transformed predictor space. For example, the value of the radial basis function (sometimes called the Gaussian kernel) is based on the magnitude of the distance between two predictor variables in the original predictor space.

Figure 377. Non-linear data transformed so as to become linearly separable

Generally speaking, a kernel function, K , is the inner product of two transformed predictors:

$$K(\vec{x}_i, \vec{x}_j) = \overrightarrow{\varphi(\vec{x}_i)} \cdot \overrightarrow{\varphi(\vec{x}_j)}$$

where φ is the transform function.

Multi-class classification models

A multi-class classification model maps predictor variables to a response variable in one of multiple classes.

In a multi-class classification model, the response, y , can be one of a number, M , of classes:

$$y \in \{y^1, \dots, y^M\}$$

A number of strategies for performing multi-class classification are supported in the SVM procedure:

Direct

Creates a single solver that attempts to solve the multi-class optimization problem directly.

One against the rest

Creates a binary classifier for each class. Each classifier separates one of the classes from the others. When classifying a point, the classifier with the largest positive score is used to indicate to which class the point belongs.

Pairwise

Creates a binary classifier for each pairwise combination of classes. If there are M classes then

$\frac{M(M-1)}{2}$ classifiers are created.

Each classifier only concerns itself with the data that belong to the pair of classes. When classifying a point, the method counts how many times the point was mapped to each class. The class with the highest occurrence is considered the one to which the point belongs.

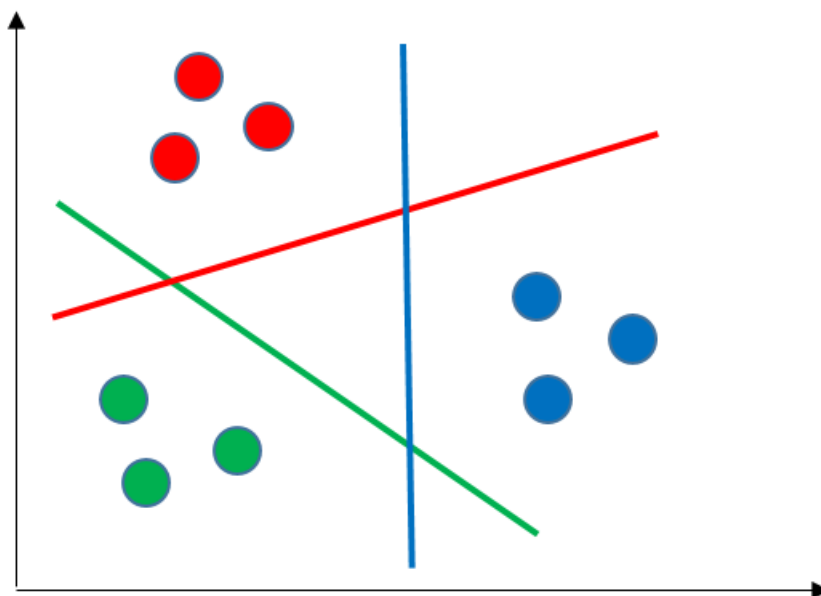
Error-Correcting Output Coding

Creates a number of binary classifiers each of which separates the classes in a particular way. Each of the classes is represented in the set of binary classifiers by a unique string of bits (a word).

When classifying a point, the method maps it to either a 1 (one) or 0 (zero) value, just as with a bit, for each of the binary classifiers. This creates a word to which the point maps. The class word that is nearest to this is considered to be the one to which the point belongs, where the *Hamming* distance is used for comparing words. See Dietterich & Bakiri, 1995 [9] for more details.

The multi-class modelling strategy is specified by setting the `MULTICLASS` option of the `MODEL` statement. For example the following statement uses the Error-Correcting Output Coding method in the SVM procedure:

```
MODEL z=x y / MULTICLASS=ECOC;
```



Single class modelling

Single class modelling is used to detect anomalies and outliers.

A single class model looks for a hyperplane in a feature space that separates out the anomalous observations from the rest. Single class modelling often uses a Gaussian kernel to determine the density function. The score function is negative for anomalous results and non-negative otherwise.

Optimisation

SVM modelling is based around linear classifiers that partition a predictor space into two.

The purpose is to find a hyperplane in predictor space that separates two types of training data such that the distance between the plane and the nearest observation of either type is as large as possible. Those observations from the training data that sit on the *fat margin* are known as support vectors (hence this method is called a Support Vector Machine).

There are various ways of solving the optimization problem posed by the model given above. The SVM procedure provides two algorithms for performing model optimisation.

1. **Iterative Single Data Algorithm** – Further details of the algorithm are described in Kecman, et al., 2005 [10].
2. **Sequential Minimal Optimization** – Further details of this optimisation algorithm are described in Platt, 1998 [11].

The type of solver used for solving the optimisation problems is specified with the `SOLVER` option of the `MODEL` statement.

Probability estimation

When classifying points, it is useful to know the probability that a point belongs to a class given its position in the predictor space.

A method for finding such posterior probabilities was pioneered by Platt, 1999 [12] and later improved by Lin, et al., 2007 [13]. In essence the probability that a point belongs to a class, P , is a logistic transformation of the classifier score, f :

$$P(y = +1 \mid \vec{x}) = \frac{1}{1 + \exp\{Af(\vec{x}) + B\}}$$

The parameters A and B are scalar values that the algorithm estimates using a maximum likelihood method on the training data.

Specifying the `PROBABILITY` option of the `MODEL` statement estimates posterior probabilities when using the SVM procedure. For example, the following statement uses *Platt Scaling* to estimate posterior probabilities:

```
MODEL z=x y / PROBABILITY=PLATT;
```

SVM Regression

The SVM procedure can fit a regression model to the provided data.

The regression model is derived in a similar way to the classification model, starting with the objective of finding a linear function that relates the predicted value for a dependent variable, y , to an independent variable, x :

$$\hat{y}_i = \vec{w}^T \vec{x}_i + b$$

The function should minimise the magnitude of w . To ensure the function fits the data, the residuals should be no greater than a specified tolerance, *epsilon*.

As with classifications, optimisation is solved using either the Sequential Minimal Optimization (SMO) algorithm or the Iterative Single Data Algorithm (ISDA) that reduce the problem into a set of smaller problems.

Use the `TYPE` option of `PROC SVM` to specify the type of problem to be solved using the SVM procedure. For example, the following statement specifies that the SVM procedure is to solve the regression problem:

```
PROC SVM TYPE=REG;
```

The size of *epsilon* used by the SVM procedure when solving a regression problem is specified with the `EPSILON` option of the `MODEL` statement. For example, the following statement specifies that the SVM procedure is to solve the regression problem using 0.5 for *epsilon*:

```
MODEL z=x y / EPSILON=0.5;
```

Alternatively, a value for the `NU` option of the `MODEL` statement can be specified. As with unary classification, the value of `NU` should be greater than 0 (zero) and less than, or equal to, 1. In this case the value of `EPSILON` is adjusted as part of the optimisation problem with the `NU` value setting an upper bound on the fraction of outliers and a lower bound on the fraction of support vectors.

Data standardisation

Data standardisation ensures that when the model parameters are derived data dimensions with larger values do not swamp data in other dimensions with smaller numerical values.

SVM models depend on the inner products between pairs of predictors. In a multidimensional predictor space, the data in some dimensions may have values of much greater magnitude than in other dimensions. In this case, an inner product aggregation between two predictors will be affected much more by the dimensions with the larger values.

If the data is not already normalized, you can use the `STANDARD` option in the `MODEL` statement to standardize the scales of the predictor variables. In this case, `PROC SVM` uses a weighted mean and standard deviation as the offset and scale estimates, respectively, for each predictor variable.

As an example the following statement specifies that the predictor variables, `temperature` and `year`, are scaled before performing the SVM modelling.

```
MODEL yield=temperature year / STANDARD=SD;
```

If the `STANDARD` option is not specified, then the predictor variables are not scaled.

Encoding categorical variables

PROC SVM supports several parameterisation encodings for categorical variables.

A variable is specified as categorical using the `CLASS` statement. For example, the following specifies that the `country` variable is categorical, and the `EFFECT` parameterisation is used to encode it:

```
CLASS country (PARAM=EFFECT);
```

Response values are always encoded using a GLM encoding.

If the model type is regression (PROC SVM has `TYPE=REG`), the response variable numeric and not in the `CLASS` statement, no encoding is performed.

Weighting and standardisation

When performing SVM modelling, observations are weighted.

The `PRIORS` option of the `MODEL` statement controls how observations are weighted. The procedure uses the following method to calculate the weights of observations.

Each class in the model has a prior probability that is either determined by the user or is based on the fraction of observations in the training data that belong to a class. A cost matrix C that transforms the vector of prior probabilities p is specified as follows:

$$p_C^* = p^T C$$

The SVM procedure normalises the set of prior probabilities using the following:

$$p_{C,k} = \frac{p_{C,k}^*}{\sum_{k=1}^K p_{C,k}^*}$$

where K is the number of classes.

The weights of each observation in a class are then normalised using the prior probability:

$$w_i^* = \frac{p_{C,k} w_i}{\sum_{j \in \text{class } k} w_j}$$

The SVM procedure uses the weights of the observations when standardizing the predictor variables:

1. L1 and L2 norms of the weights are calculated:

$$v_1 = \sum_i w_i^*, v_2 = \sum_i (w_i^*)^2$$

2. The offset coordinate is calculated:

$$\mu = \frac{\sum_i w_i^* x_i}{\sum_i w_i^*}$$

3. The scaling vector is calculated:

$$\sigma = \sqrt{\frac{v_1}{v_1^2 - v_2} \sum_i w_i^* (x_i - \mu)}$$

4. The predictor coordinates are updated:

$$\forall i, x_i^* = \frac{x_i - \mu}{\sigma}$$

To specify the prior probabilities of classes, use the `MODEL` statement, with `PRIORS=CUSTOM`. For example, the following statement sets prior probabilities for classes 'red', 'green' and 'blue':

```
MODEL z=x y / PRIORS=CUSTOM ('red' = 0.5 'green' = 0.3 'blue' = 0.2);
```

To specify the cost matrix (or part of it), use the `MODEL` statement, with `PRIORS=COST`. For example, the following statement sets the cost of incorrectly classifying a sample that belongs to the 'red' class as a member of the 'blue' class:

```
MODEL z=x y / PRIORS=COST (('red' 'blue') = 2);
```

Using the SVM procedure

This example shows how to use `PROC SVM` to analyse data and predict results.

This example uses the publicly-available Iris dataset [14]. The Iris dataset consists of measurements of the widths and lengths of the petals and sepals of three species of Iris.

The model defines the petal length, `p_length`, and the petal width, `p_width`, as predictors of the class `species`. It uses a multi-class 'one against the rest' classification system, and a Gaussian kernel. This example specifies `BOXCONSTRAINT` and `DELTAGRADTOL` values, and allows a maximum of 10,000 iterations. It uses the Platt scaling method for estimating posterior probabilities for each class, and outputs the model statistics and a variety of plots showing how the data is distributed.

```
PROC SVM
  DATA = iris
  PLOTS = (CONTOUR (OBS = OUTLINE UNPACK) HEATMAP (OBS))
  TYPE = CLASS;
  CLASS species;
  MODEL species = p_length p_width /
    BOXCONSTRAINT = 1.0
    DELTAGRADTOL = 0.001
    DETAILS (OUTPUTSTATS)
    KERNEL = GAUSSIAN (RBFALLOFF = 1.0)
    MAXITERS = 10000
    MULTICLASS = ONEVREST
    PROBABILITY = PLATT;
RUN;
```

This example produces the following output.

Predictor Variable Standardization

The Predictor Variable Standardization table records the data standardisation applied in `PROC SVM`. In this case, no standardisation was applied, as `STANDARD=SD` was not specified in the `MODEL` statement.

Standardization applied: none		
Statistic	p_length	p_width
Offset	0.00000	0.00000
Scale	1.00000	1.00000

Fit Summary

The Fit Summary table summarises the SVM model created in `PROC SVM`. This was specified as a multi-class, one against the rest problem. The data includes three species of iris, so there are three classes (setosa, versicolor and virginica). The model needs to solve three problems:

- Is this observation a member of the setosa class or not? This is problem 0.
- Is this observation a member of the versicolor class or not? This is problem 1.
- Is this observation a member of the virginica class or not? This is problem 2.

The fit table (below) shows some general information about the model parameters, then shows the details for each of these problems separately. You can see that the models converged for all three problems, problem 0 after 298 iterations, problem 1 after 105 iterations and problem 2 after 134 iterations.

You can specify `DETAILS (MODELPARAMS)` in the `MODEL` statement to see more information about the model parameters.

Attribute	Value
SVM Type	Classification
Solver Type	SMO
Number of Observations	150
Number of Predictor Dimensions	2
Number of Classes	3
Classification Accuracy	0.96667
Kernel Type	Gaussian
Gaussian Falloff	1.00000
Regularization Parameter	1.00000
Maximum Iterations	10000
Delta Gradient Tolerance	0.00100
Feasibility Gap Tolerance	0.00000
KKT Violation Tolerance	0.00000
Number of Problems	3
Problem	0
Sample Count	150
Converged	Yes
Iteration Count	298
Support Vector Count	13
Largest KKT Violation	0.00005
Largest Delta Gradient	0.00008
Feasibility Gap	0.00005
Primal Objective Value	2.16860
Dual Objective Value	2.16843
$ w ^2$	4.29970
Sum of dual variable values	4.31828
Sum of slack variable values	0.01876
Problem	1
Sample Count	150
Converged	Yes
Iteration Count	105
Support Vector Count	26
Largest KKT Violation	0.00001
Largest Delta Gradient	0.00005
Feasibility Gap	0.00000
Primal Objective Value	17.50874
Dual Objective Value	17.50871
$ w ^2$	9.66152
Sum of dual variable values	22.33947
Sum of slack variable values	12.67798
Problem	2
Sample Count	150
Converged	Yes
Iteration Count	134
Support Vector Count	27
Largest KKT Violation	0.01580
Largest Delta Gradient	0.00056
Feasibility Gap	0.00350
Primal Objective Value	17.73392
Dual Objective Value	17.66840
$ w ^2$	10.35378
Sum of dual variable values	22.84529
Sum of slack variable values	12.55703

Output Stats

The Output Stats table (extract below) lists the observations in the dataset (in this case, the petal length, the petal width and the actual species) and the predicted class (the species that the model predicts this petal has come from).

The final three columns (labelled *setosa*, *versicolor* and *virginica*) give the posterior probabilities for each observation (the probabilities for each class that the observation is actually a member of that class). So if the model is a good fit, the probability will be higher for the actual class than for the other two classes. In most cases, the predicted class based on the probabilities is the same as the actual class, but there are one or two discrepancies where the probability-based predicted class is not the same as the actual class, for example, observation 71, where a versicolor iris has been predicted to be virginica by the model, and observation 107, where a virginica iris has been predicted to be versicolor. But even in these cases, the posterior probabilities for the actual and predicted classes are quite close in value.

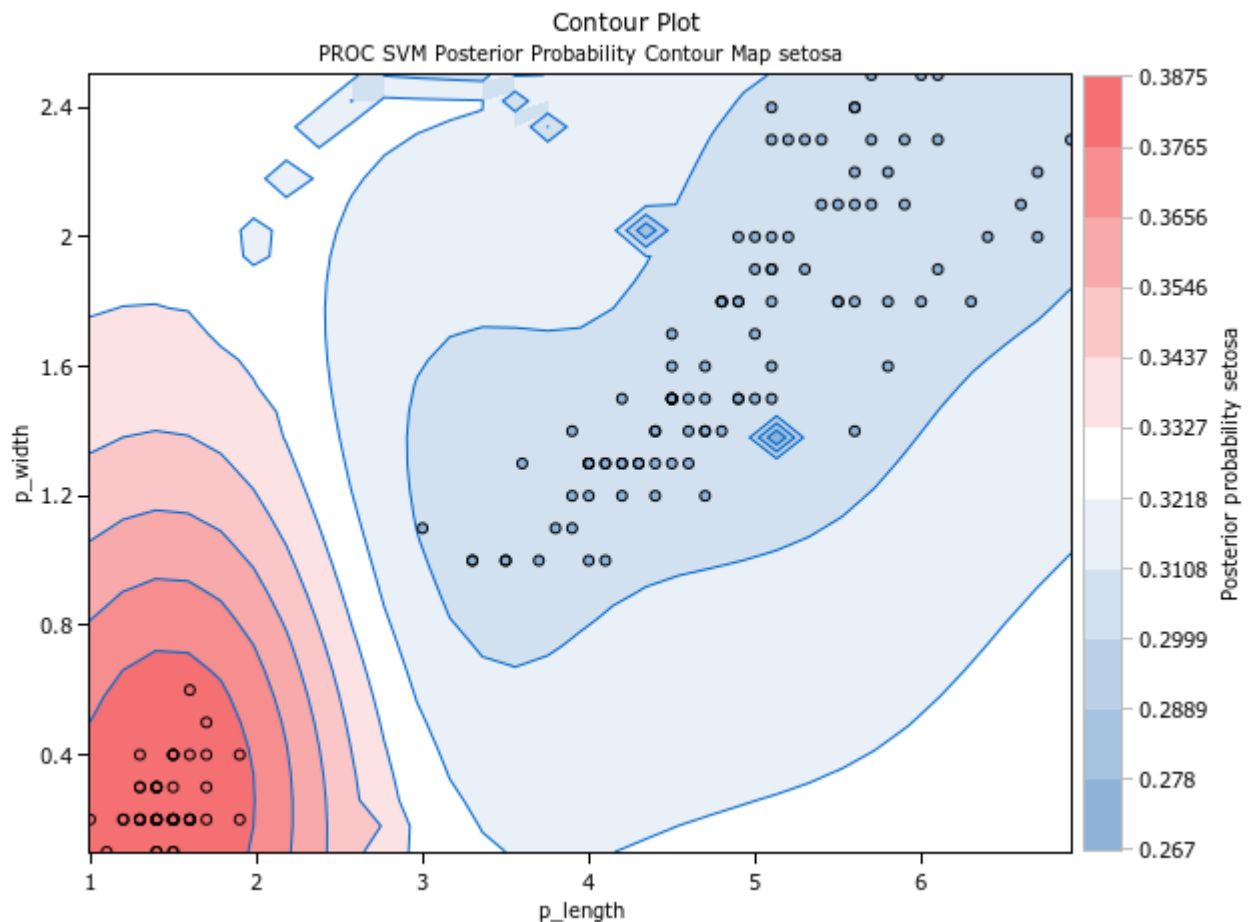
Observation Index	p_length	p_width	species	Predicted Class	setosa	versicolor	virginica
1	1.40000	0.20000	setosa	setosa	0.38718	0.30645	0.30637
2	1.40000	0.20000	setosa	setosa	0.38718	0.30645	0.30637
3	1.30000	0.20000	setosa	setosa	0.38627	0.30712	0.30661
4	1.50000	0.20000	setosa	setosa	0.38732	0.30620	0.30648
5	1.40000	0.30000	setosa	setosa	0.38747	0.30624	0.30629
6	1.70000	0.40000	setosa	setosa	0.38463	0.30766	0.30771
7	1.40000	0.30000	setosa	setosa	0.38747	0.30624	0.30629
8	1.50000	0.20000	setosa	setosa	0.38732	0.30620	0.30648
9	1.40000	0.20000	setosa	setosa	0.38718	0.30645	0.30637
10	1.50000	0.10000	setosa	setosa	0.38582	0.30694	0.30724
. . .							
65	3.60000	1.30000	versicolor	versicolor	0.30608	0.38650	0.30742
66	4.40000	1.40000	versicolor	versicolor	0.30779	0.39232	0.29988
67	4.50000	1.50000	versicolor	versicolor	0.30754	0.38317	0.30930
68	4.10000	1.00000	versicolor	versicolor	0.30912	0.39579	0.29508
69	4.50000	1.50000	versicolor	versicolor	0.30754	0.38317	0.30930
70	3.90000	1.10000	versicolor	versicolor	0.30751	0.39361	0.29888
71	4.80000	1.80000	versicolor	virginica	0.30655	0.33867	0.35478
72	4.00000	1.30000	versicolor	versicolor	0.30766	0.39497	0.29737
73	4.90000	1.50000	versicolor	versicolor	0.30623	0.35802	0.33575
74	4.70000	1.20000	versicolor	versicolor	0.30833	0.38881	0.30287
. . .							
101	6.00000	2.50000	virginica	virginica	0.31001	0.30837	0.38161
102	5.10000	1.90000	virginica	virginica	0.30542	0.31114	0.38344
103	5.90000	2.10000	virginica	virginica	0.30702	0.29881	0.39417
104	5.60000	1.80000	virginica	virginica	0.30573	0.29909	0.39518
105	5.80000	2.20000	virginica	virginica	0.30736	0.29769	0.39495
106	6.60000	2.10000	virginica	virginica	0.30736	0.31212	0.38053
107	4.50000	1.70000	virginica	versicolor	0.30811	0.36646	0.32543
108	6.30000	1.80000	virginica	virginica	0.30861	0.30738	0.38401
109	5.80000	1.80000	virginica	virginica	0.30659	0.29882	0.39459
110	6.10000	2.50000	virginica	virginica	0.30980	0.31016	0.38004
. . .							

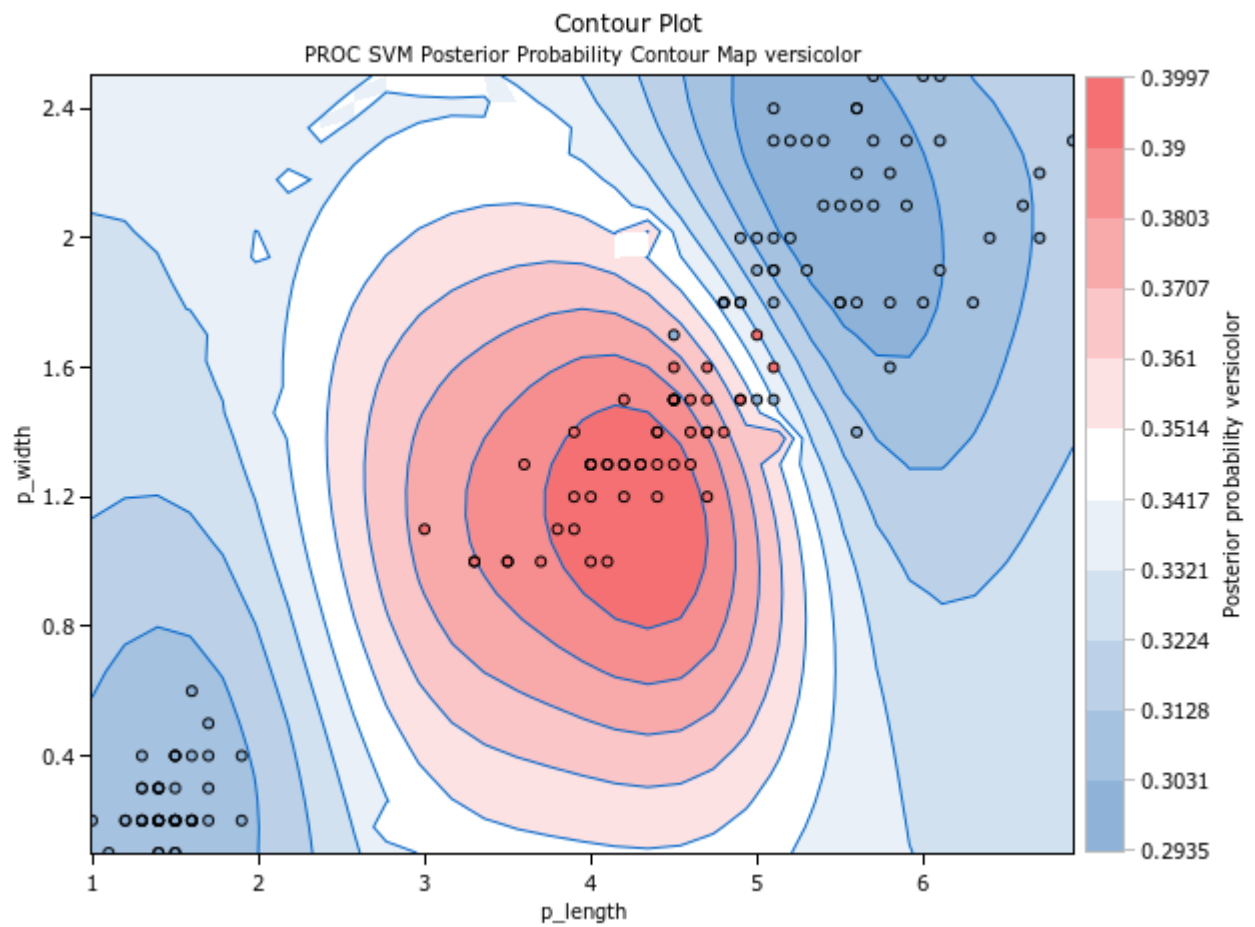
Contour Plots

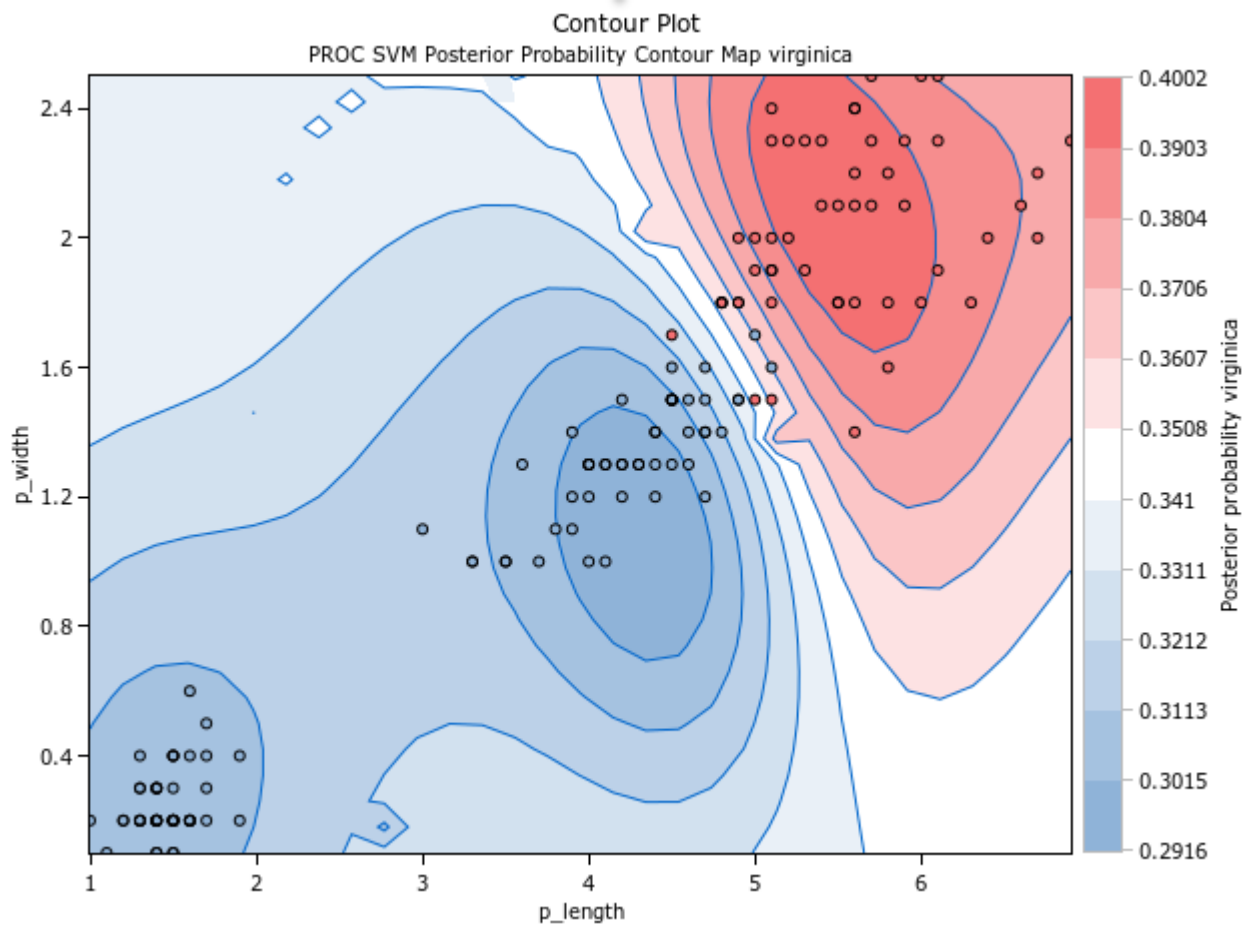
The contour plots show the distribution of the values of each pair of predictor variables, and the posterior probability that each pair is a member of one of the classes in the model. There is one contour plot for each class. The data points shown on each plot are the same, but the posterior probabilities for each data point are different for each class.

In each plot, the dark red areas show values that are very likely to belong to that class, graduating through lighter red, white, light blue and dark blue for observations that are less and less likely to belong to that class. Note that this kind of plot can only be produced for classification models that have exactly two predictor variables.

In the second of the three plots, the red dots in the blue areas represent the versicolor observations that have been predicted to be virginica. In the third plot, the red dots in the blue areas represent the virginica observations that have been predicted to be versicolor.

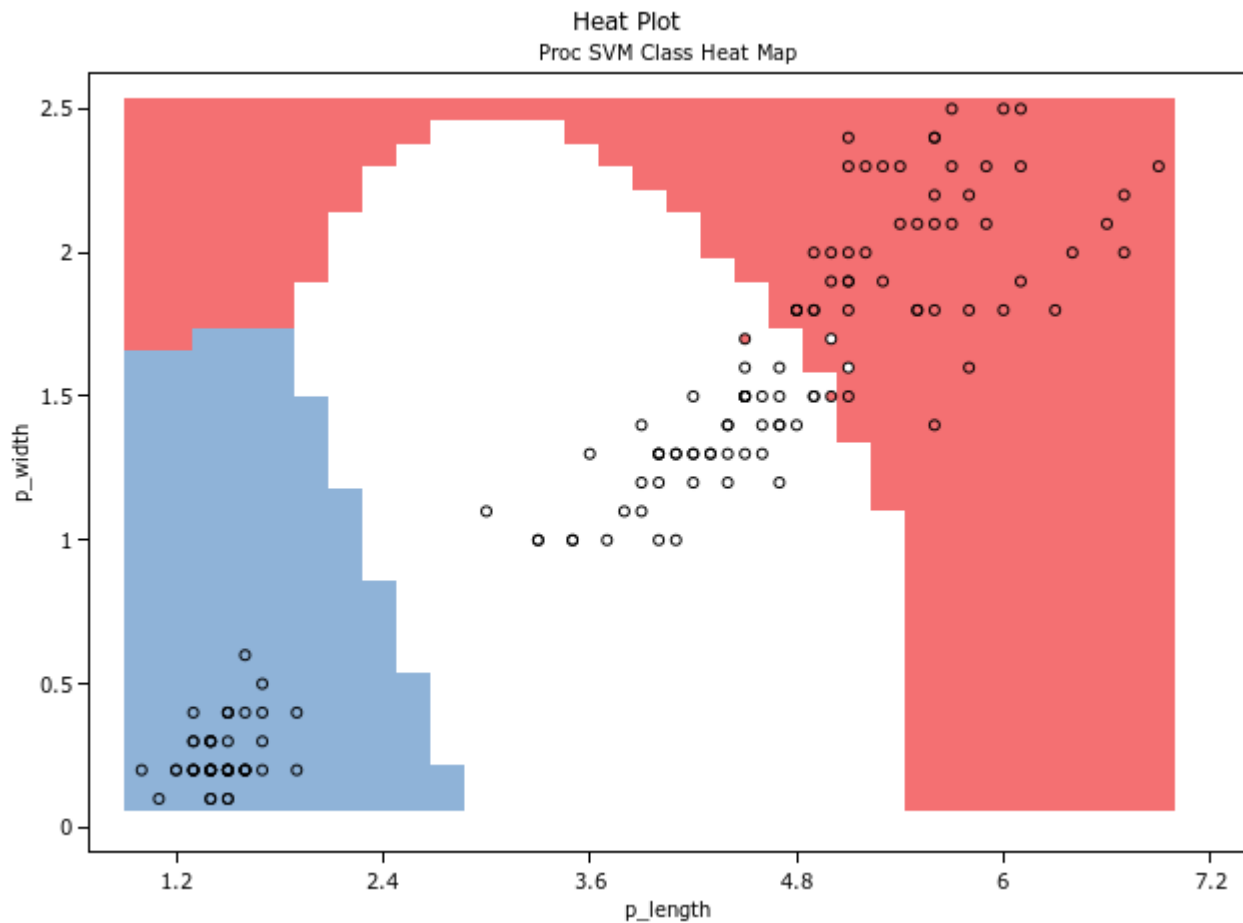






Heat plot

The heat plot shows the distribution of the values of each pair of predictor variables, and the class for each pair as predicted by the model. In this plot, the setosa values are shown in blue, versicolor in white and virginica in red. The white spots in the red area are the versicolor observations that were misclassified as virginica and the red spots in the white area are the virginica observations that were misclassified as versicolor.



Scoring another dataset

This model could then be used on another dataset to predict classifications from the predictor variables. To do this, use the `SCORE` statement.

SVM procedure reference

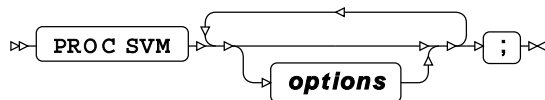
Describes the syntax and options for `PROC SVM` and its contained statements.

<code>PROC SVM</code> ↗	3364
Configures a <i>Support Vector Machine</i> to analyse the specified dataset.	
<code>BY</code> ↗	3369
Groups the observations using one or more specified variables.	
<code>CLASS</code> ↗	3369
Specifies the class variables, and, optionally, how they are handled.	
<code>FREQ</code> ↗	3372
Specifies a variable in the dataset that defines the relative frequency of each observation.	

ID ↗	3372
Identifies the relevant observations in the output by using one or more specified variable names.	
KERNELINNER ↗	3373
Specifies the inner part of the custom kernel function to use to transform the data into a linear space before deriving the SVM model.	
KERNELOUTER ↗	3373
Specifies the outer part of the custom kernel function to use to transform the data into a linear space before deriving the SVM model.	
MODEL ↗	3374
Specifies the response variable, the predictor variables and the options to be used for the SVM model.	
OUTPUT ↗	3383
Creates a new dataset containing the input observations and the predicted values calculated by the SVM model.	
SCORE ↗	3385
Uses the current SVM model to score or classify the data in the specified dataset.	
WEIGHT ↗	3387
Specifies a variable in the input dataset giving the prior weight associated with each observation.	

PROC SVM

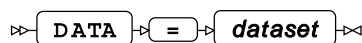
Configures a *Support Vector Machine* to analyse the specified dataset.



Options

The following *options* are available:

DATA



Specifies the training dataset used by `PROC SVM` to construct the SVM model.

If neither the `DATA` nor the `INMODEL` options are specified, the most recently-created dataset is used as the training dataset. The `DATA` option cannot be specified if the `INMODEL` option is also specified.

INMODEL



Specifies the location of a previously-saved, serialised SVM model to be used to score another dataset. The dataset to be scored is specified using the `SCORE` statement.

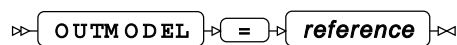
If the serialised model includes groupings specified in a `BY` statement, the groupings are included in the serialisation and are applied automatically. The groupings in the serialised model override any groupings that are explicitly specified in a `BY` statement.

The `INMODEL` option cannot be specified if the `DATA` option is also specified.

reference

The serialised model name is specified as *library-name.item-name*, where *library-name* is a library reference, and *item-name* is the item contained by the library. If *library-name* is not specified, the default `WORK` library is used.

OUTMODEL



Specifies that the SVM model created from the training dataset is saved as a serialised model in the specified location. This serialised model can later be used to score another dataset.

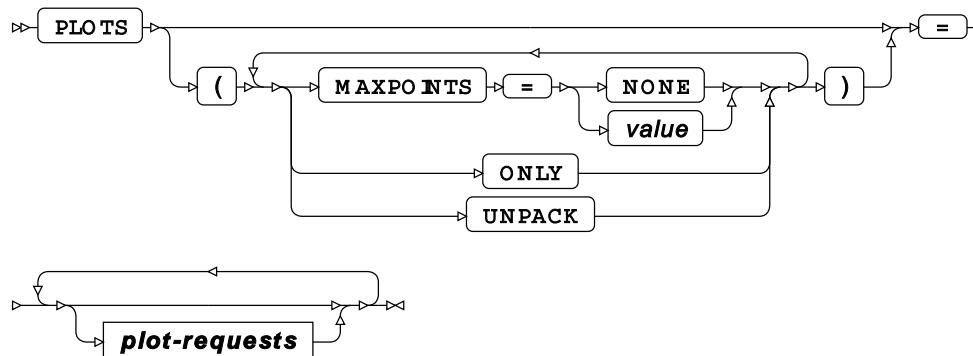
If the model being saved includes any groupings specified in a `BY` statement, the groupings are saved with the serialised model.

reference

The serialised model name is specified as *library-name.item-name*, where *library-name* is a library reference, and *item-name* is the item contained by the library. If *library-name* is not specified, the default `WORK` library is used.

PLOTS

Specifies which plots are produced through the ODS Graphics framework.



MAXPOINTS

Specifies the maximum number of elements allowed in a plot. Any plot containing more than the specified number of elements is not output. The default value is 5000 elements.

NONE

Specifies that the cut-off value is ignored.

value

Specifies the maximum number of elements in a plot.

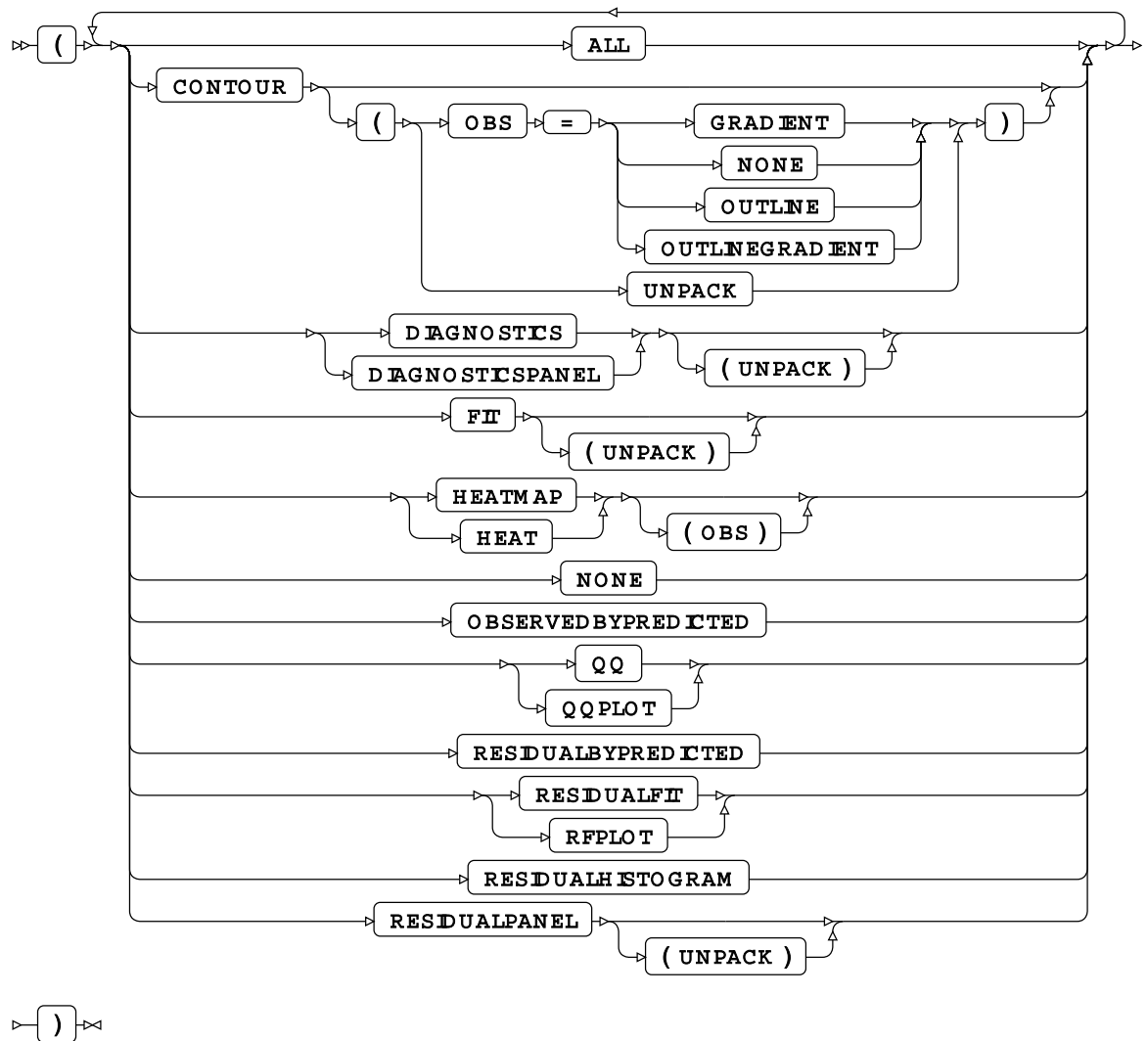
ONLY

Specifies that default plots are not produced unless specifically requested.

UNPACK

Specifies that the plots are displayed individually. By default, the plots are displayed in a panel.

plot-requests



ALL

Specifies that all plots appropriate for the model are produced.

CONTOUR

Produces a set of contour plots with an optional overlaid scatter plot.

If `TYPE=CLASS` is specified in the `PROC SVM` statement, each plot displays the posterior probability for a class in the model.

If `TYPE=REG` is specified in the `PROC SVM` statement, a single contour plot of the response variable is produced.

The `CONTOUR` option requires a model with two predictor variables, and, if `TYPE=CLASS` is specified in the `PROC SVM` statement, `PROBABILITY=PLATT` must be specified in the `MODEL` statement.

OBS

Specifies that observations are overlaid on top of the plot. The `OBS` option is only valid if `TYPE=REG` is specified in the `PROC SVM` statement. The following observation types are supported:

GRADIENT

Specifies that observations are displayed as circles with a coloured fill.

NONE

Specifies that observations are not displayed. This is the default.

OUTLINE

Specifies that observations are displayed as circles with a border but no fill.

OUTLINEGRADIENT

Specifies that observations are displayed as circles with both a border and a coloured fill.

UNPACK

Specifies that the plots should be individually displayed.

DIAGNOSTICS

Produces a summary set of regression fit diagnostics consisting of:

- Residuals against predicted values.
- Histogram of the residuals.
- Normal quantile plot of the residuals.
- Residual fit plot.
- Residual against predictor plots.
- Observed against predicted plot.

The `DIAGNOSTICS` option is only valid if `TYPE=REG` is specified in the `PROC SVM` statement.

UNPACK

Specifies that the plots should be individually displayed.

FIT

Produces a set of fit plots for models with one predictor variable.

If `TYPE=CLASS` is specified, each plot displays the posterior probability for a class in the model.

If `TYPE=REG` is specified, a single scatter plot of the predicted response is produced.

The `FIT` option requires a model with one predictor variable, and, if `TYPE=CLASS` is specified in the `PROC SVM` statement, `PROBABILITY=PLATT` must be specified in the `MODEL` statement.

UNPACK

Specifies that fit plots are individually displayed. This option only valid if `TYPE=CLASS` is specified

HEATMAP

Produces a heat map plot of the predicted class against predictor values overlaid with a scatter plot of the data for models with two predictor variables.

The `HEATMAP` option requires a model with two predictor variables, and `TYPE=CLASS` specified in the `PROC SVM` statement.

OBS

Specifies that the observations should be overlaid on top of the plot.

NONE

Specifies that no plot output is produced.

OBSERVEDBYPREDICTED

Produces a scatter plot of the observed response variable against the predicted response values.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

QQ

Produces a normal quantile plot of the residuals.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

RESIDUALBYPREDICTED

Produces a scatter plot of the residuals against the predicted values.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

RESIDUALFIT

Produces a pair of quantile plots showing the centred fit and the residuals.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

RESIDUALHISTOGRAM

Produces a histogram of the residuals.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

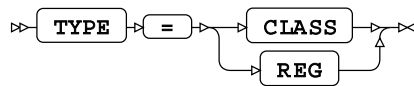
RESIDUALPANEL

Produces a set of scatter plots. Each plot displays the residuals against predictor coordinate values.

This option requires that `TYPE=REG` is specified in the `PROC SVM` statement.

UNPACK

Specifies that the plots are individually displayed.

TYPE

Specifies the type of problem to solve.

CLASS

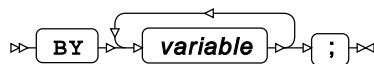
Specifies that a classification problem is to be solved. This is the default `TYPE`.

REG

Specifies that a regression problem is to be solved.

BY

Groups the observations using one or more specified variables.

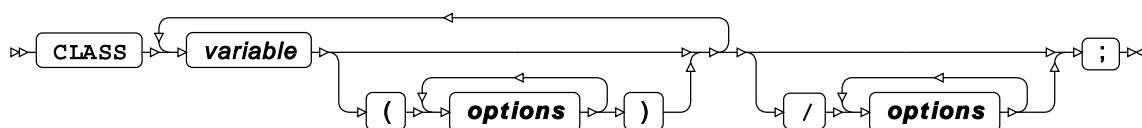


The specified variable or variables are used to separate the input data into groups. `PROC SVM` generates a separate model from the data in each group. If the `BY` statement is included, the input dataset must be pre-sorted on the specified variable or variables.

If a variable is specified as a predictor variable or response variable in the `MODEL` statement, it cannot also be specified in the `BY` statement. If `PROC SVM` includes the `INMODEL` option to use a pre-existing serialised model, the `BY` options, if any, are derived from the serialised model, and any options specified here are ignored.

CLASS

Specifies the class variables, and, optionally, how they are handled.



A class (or categorical) variable is a variable that can take one of a limited number of values.

The options, if present, define the way the class variables are handled in the model. Options can be specific to a single variable or can be global. Options in brackets after a variable name are specific to that variable. Options specified after the forward slash are global options and apply to all class variables. Variable-specific options override global options. Unless otherwise specified, all options can be variable-specific or global.

The response variable always uses the ascending sort order, the `FORMATTED` ordering method, `GLM` encoding and the `LAST` reference level.

If present, the `CLASS` statement or statements must be located before the `MODEL` statement.

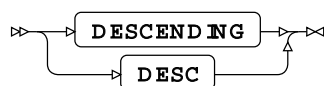
variable

A class variable in the dataset.

Options

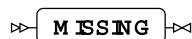
The following *options* are available. Unless otherwise stated, all options can be applied to a single variable, or globally to all variables.

DESCENDING



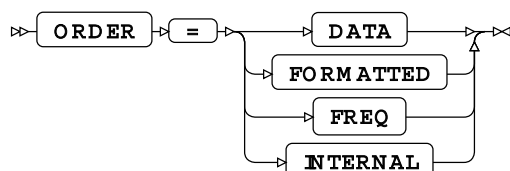
Specifies that the variable values are sorted in descending order. If not specified, the variable values are sorted in ascending order. The way in which the variable values are ordered is specified using the `ORDER` option.

MISSING



Specifies that a level is created for missing values and that observations containing missing values are retained. If not specified, observations with missing values are discarded.

ORDER



Specifies the ordering to use for variable values. Variable values are ordered in ascending order unless the `DESCENDING` option is also specified.

For numeric variables, the default ordering is `INTERNAL`. For string variables, the default ordering is `FORMATTED`.

DATA

The variable values are ordered in the order in which the values of the variable first occur in the data.

FORMATTED

The variable has a user-defined format applied, and the variable values are ordered using the variable format value.

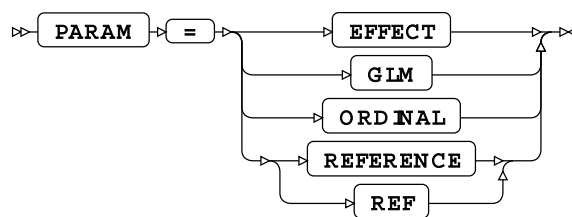
FREQ

The variable values are ordered based on the frequency count of the values in the input dataset. Values that occur more frequently appear earlier in the ordering than values that occur less frequently.

INTERNAL

The variable is unformatted, and the variable values are ordered by the raw value.

PARAM



Specifies the parameterisation encoding for the variable.

Response variables always use GLM encoding. The default encoding for effect (or predictor) variables is GLM encoding.

EFFECT

Specifies that the variable is encoded using effect encoding.

GLM

Specifies that the variable is encoded using GLM encoding.

This value can only be specified as a global option and not for an individual variable.

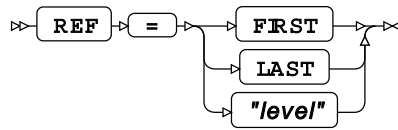
ORDINAL

Specifies that the variable is encoded using ordinal encoding.

REFERENCE

Specifies that the variable is encoded using reference encoding.

REF



Specifies the reference level to use for a class variable.

FIRST

The reference level is the first ordered level.

LAST

The reference level is the last ordered level. This is the default reference level.

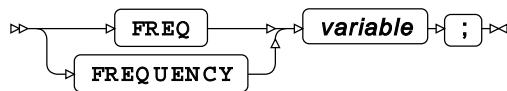
"level"

Specifies a value to use as the reference level. The specified value must be a valid value for the class variable.

This value can only be specified for individual variables, not globally.

FREQ

Specifies a variable in the dataset that defines the relative frequency of each observation.



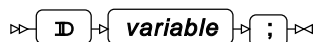
If a frequency variable is defined, the existing weight of each observation is multiplied by its associated frequency to get a new weight.

variable

The variable in the dataset that contains the frequency value for each observation.

ID

Identifies the relevant observations in the output by using one or more specified variable names.



KERNELINNER

Specifies the inner part of the custom kernel function to use to transform the data into a linear space before deriving the SVM model.

```
» KERNELINNER ; «  
  
» transform-statements «  
  
» ENDKERNELINNER ; «
```

The inner kernel function maps the numeric inputs `_x1_` and `_x2_` to a single numeric output `_inner_`. The inner kernel function is paired with an outer kernel function which maps the sum of the `_inner_` values defined in the inner kernel to a single output `_k_`.

An inner kernel function and the corresponding outer kernel function must be defined if `KERNEL = CUSTOM` is specified in the `MODEL` statement.

Example

This example shows a simple inner kernel function and the corresponding outer kernel function for a Gaussian kernel transformation.

```
KERNELINNER; x1 = _x1_; x2 = _x2_; d = x1 - x2;  
    _inner_ = d * d;  
ENDKERNELINNER;  
KERNELOUTER; a = _agg_;  
    _k_ = exp(-a);  
ENDKERNELOUTER;
```

KERNELOUTER

Specifies the outer part of the custom kernel function to use to transform the data into a linear space before deriving the SVM model.

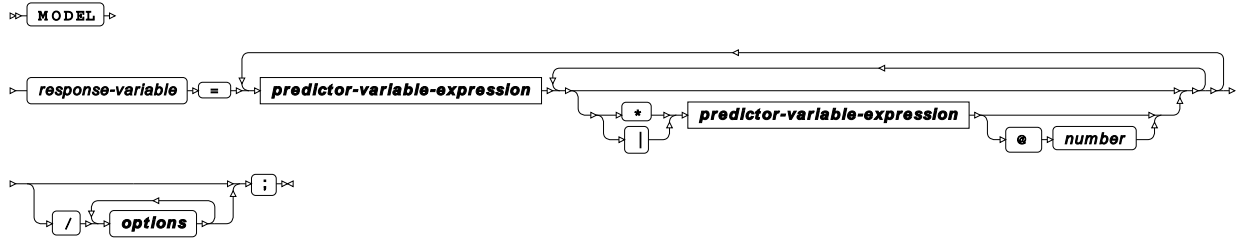
```
» KERNELOUTER ; «  
  
» transform-statements «  
  
» ENDKERNELOUTER ; «
```

The outer kernel function maps the numeric input `_agg_` (the aggregate of the `_inner_` values defined in the inner kernel function) to a single numeric output `_k_`.

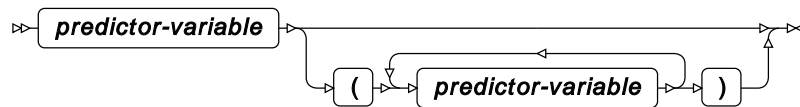
An inner kernel function and the corresponding outer kernel function must be defined if `KERNEL = CUSTOM` is specified in the `MODEL` statement.

MODEL

Specifies the response variable, the predictor variables and the options to be used for the SVM model.



predictor-variable-expression



Each PROC SVM must include exactly one MODEL statement, unless PROC SVM specifies a previously-saved, serialised model using the INMODEL option. In that case, the model definition specified by INMODEL is used.

response-variable

Specifies the response variable in the input dataset.

predictor-variable

Specifies a variable in the input dataset to include in the model as an predictor variable, and optionally, specifies how it is combined with other variables to derive new predictor variables to include in the model.

The variables can be combined in the same way as variables in other regression procedures:

- * Include a new variable that is the product of the specified variables.
- For example, `var1*var2` defines an effect variable that is the product of `var1` and `var2`.
- | Include each specified variable. Also include new variables from the products of each possible combination of two or more of the specified variables .
- For example, `var1 | var2 | var3` defines the effect variables `var1`, `var2`, `var3`, `var1*var2`, `var1*var3`, `var2*var3` and `var1*var2*var3`.

- @ When combining multiple variables to make a new variable, include no more than the specified number of variables in each combination.

For example, `var1 | var2 | var3@2` defines the effect variables `var1`, `var2`, `var3`, `var1*var2`, `var1*var3` and `var2*var3`, but not `var1*var2*var3`.

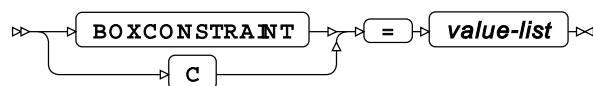
- () The variable outside the brackets is nested on all possible discrete values of the variable inside the brackets, and each nesting defines a new effect variable. The variable inside the brackets must be a discrete variable specified in a `CLASS` statement.

For example, if `var2` has values A, B and C, `var1 (var2)` defines the effect variables `var1 (A)`, `var1 (B)` and `var1 (C)`.

Options

The following *options* are available:

BOXCONSTRAINT



Specifies the box constraints (or regularization parameters) to be used when solving the optimization problem. Multiple values can be specified in a space-separated list, delimited by a single pair of brackets, for example, `BOXCONSTRAINT = (0.1, 1.0, 10.0)`. If multiple values are specified then all values are tried and the value that gives the best fit model is selected.

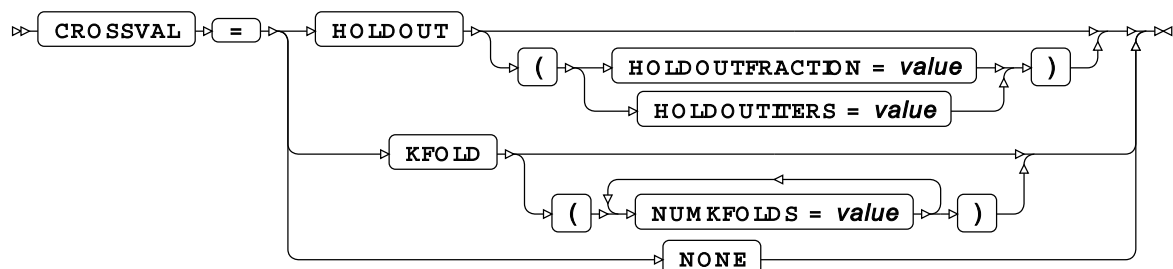
The box constraint values must be positive. The default is `BOXCONSTRAINT = (0.1, 1.0, 10.0)`.

CLIPDUALS



Specifies whether variables in the dual optimization problem should be clipped to either zero or the box constraint, if considered to be close enough to either.

CROSSVAL



Specifies the cross validation options to use in the SVM model.

HOLDOUT

The procedure randomly reserves a portion of the data for validation.

HOLDOUTFRACTION

Specifies the fraction of the training data to reserve for validation when using the **HOLDOUT** option. The default fraction is 0.1.

HOLDOUTITERS

Specifies the number of holdout models to evaluate. The default is 1.

KFOLD

The procedure randomly partitions the data into number sets.

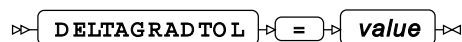
NUMKFOLDS

Specifies how many partitions are used when using the **KFOLD** option. The default is 10.

NONE

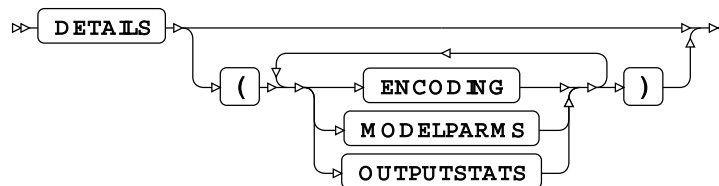
Do not use cross-validation. This is the default value.

DELTAGRADTOL



Specifies the tolerance value to use to determine convergence. The model will converge when movements in the dual space are less than or equal to this value. If this value is 0 (zero) the gradient tolerance method is not used to check for convergence. The default value is 1×10^{-3} if **SOLVER=SMO** is specified (by default the SMO solver uses the gradient tolerance method to check for convergence) and 0 if **SOLVER=ISDA** is specified (by default the ISDA solver does not use the gradient tolerance method to check for convergence).

DETAILS



Specifies which tables to display.

ENCODING

Displays the class to problem design matrix for classification models.

MODELPARMS

Displays the parameters of the SVM model.

If **KERNEL=LINEAR** then the coefficients of the separating hyperplane (including the bias term) is displayed for each problem in the SVM model.

If a non-linear kernel is used (`KERNEL` is not `LINEAR`), then each support vector of each problem in the SVM model is displayed complete with its dual variable value and standardized predictor coordinates.

OUTPUTSTATS

For SVM classification modelling (`TYPE=CLASS` in the `PROC SVM` statement), displays the classification modelling results for each observation provided in the input dataset. The following fields are included:

- Independent variable coordinates
- Actual dependent value
- Predicted dependent value
- Posterior probabilities (if probability estimation is performed)

For SVM regression modelling (`TYPE=REG` in the `PROC SVM` statement), displays the regression modelling results for each observation provided in the input dataset. The following fields are included:

- Independent variable coordinates
- Actual dependent value
- Predicted dependent value
- Residual value (if the `RESIDUAL` option is set)

EPSILON

⇒ **EPSILON** ⇒ = ⇒ *value-list* ⇒

For SVM regression modelling (`TYPE=REG` in the `PROC SVM` statement), specifies the tolerance for the residuals. The default value is 1.

Multiple values can be specified in a space-separated list, delimited by a single pair of brackets, for example, `EPSILON = (1.0, 1.1, 1.2)`. If multiple values are specified then all values are tried and the value that gives the best fit model is selected.

GAPTOL

⇒ **GAPTOL** ⇒ = ⇒ *value* ⇒

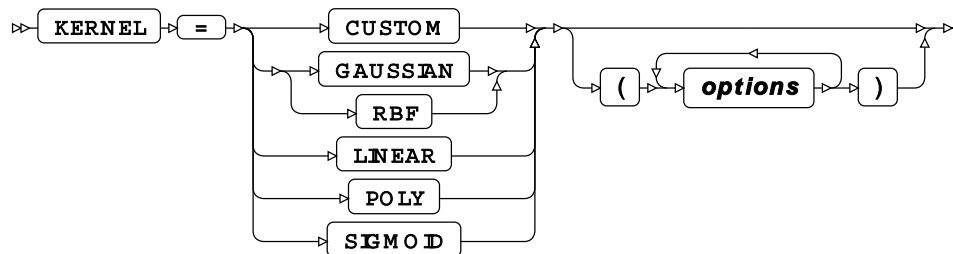
Specifies the tolerance value to use to determine convergence. The model will converge when the difference between the dual and primal objective functions is less than or equal to this value. If this value is 0, the gap tolerance method is not used to check for convergence. The default is 0 (zero), that is, the gap tolerance method is not used to check for convergence.

INITDUALS

⇒ **INITDUALS** ⇒ = ⇒ *variable* ⇒

Specifies a variable in the input dataset that represents the set of initial values for each of the variables in the dual optimization problem. In SVM classification the values should be non-negative.

KERNEL



Specifies what type of kernel function is used by the procedure.

CUSTOM

Specifies that a custom kernel function is used. If this option is selected, then the `KERNELINNER` and `KERNEL OUTER` statements must be used to define the custom kernel.

GAUSSIAN

Specifies that a radial basis function (RBF) is used. This is the default kernel function.

LINEAR

Specifies that a linear kernel is used.

POLY

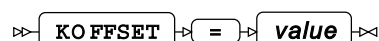
Specifies that an inhomogeneous polynomial kernel is used.

SIGMOID

Specifies that a sigmoid kernel is used.

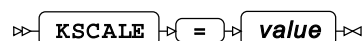
Options

KOFFSET



Specifies that the SVM procedure should add the specified number to each element of the Gram matrix. The default value is 0 (zero).

KSCALE



Specifies that the SVM procedure should scale each element in the Gram matrix by the specified number. The default value is 1.

POLYOFFSET

▷▷ POLYOFFSET ▷=▷ value-list ▷◁

Specifies a set of offset values to use if a polynomial kernel is used. The default is a single value of 0 (zero) (that is, a homogeneous polynomial).

POLYORDER

▷▷ POLYORDER ▷=▷ value-list ▷◁

Specifies a set of order values to use if a polynomial kernel is used. The default is a single value of 3.

RBFFALLOFF

▷▷ RBFFALLOFF ▷=▷ value-list ▷◁

Specifies a set of falloff values to use if a Gaussian, or radial basis function, kernel is used. The default is a single value of 1.

SIGOFFSET

▷▷ SIGOFFSET ▷=▷ value-list ▷◁

Specifies a set of offset values to use if a sigmoid kernel is used. The default is a single value of 1.

SIGSCALE

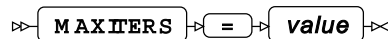
▷▷ SIGSCALE ▷=▷ value-list ▷◁

Specifies a set of scale values to use if a sigmoid kernel is used. The default is a single value of 1.

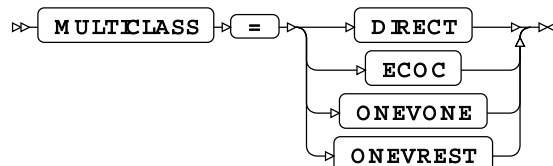
KKTTOL

▷▷ KKTTOL ▷=▷ value ▷◁

Specifies the tolerance value to use to determine convergence. The model will converge when the worst violation of the Karush-Kuhn-Tucker complementarity conditions is less than or equal to this value. If this value is 0 (zero) the Karush-Kuhn-Tucker violation value is not used to check for convergence. The default value is 0 if SOLVER=SMO is specified (by default the SMO solver does not use Karush-Kuhn-Tucker complementarity conditions to check for convergence) and 0.001 if SOLVER=ISDA is specified (by default the ISDA solver uses Karush-Kuhn-Tucker complementarity conditions to check for convergence) .

MAXITERS

Specifies the maximum number of iterations allowed for the optimization procedure. The default value is 1.0E6.

MULTICLASS

Specifies the method used by the procedure for multi-class SVM modelling. Valid values are as follows:

DIRECT

Specifies that the SVM procedure uses a direct approach to solving the multi-class SVM classification problem.

ECOC

Specifies that the SVM procedure uses the Error-Correcting Output Codes method.

ONEVONE

Specifies that the SVM procedure uses the one-against-one method. This is the default method.

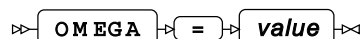
ONEVREST

Specifies that the SVM procedure uses the one-versus-the-rest method.

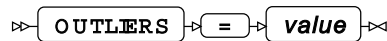
NU

Specifies the one class learning parameter **NU**. This value should be a real numeric value that is greater than 0 (zero) and less than or equal to 1. The default value is 0.5. If this option is specified with SVM regression modelling then the v-SVR solver is used.

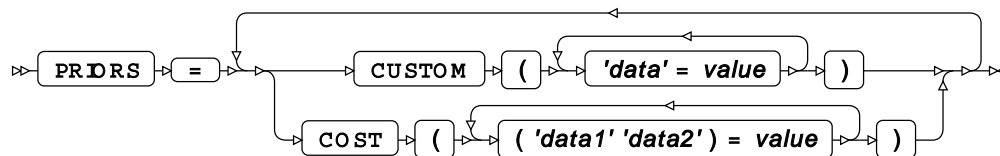
Multiple values can be specified in a space-separated list, delimited by a single pair of brackets, for example, **NU** = (0.1, 0.3, 0.5). If multiple values are specified then all values are tried and the value that gives the best fit model is selected.

OMEGA

If **SOLVER=ISDA**, specifies the value of **OMEGA** to use for the step size. The default value is 0.1.

OUTLIERS

The proportion of the observations that are regarded as outliers, to ensure robust modelling. The default value is 0 (zero), that is, there are no outliers. If specified, $0 < \text{value} < 1$.

PRIORS

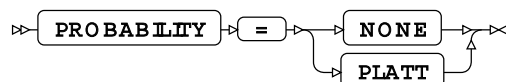
Specifies that the method should use prior probabilities when calculating the weights of the observations. Valid values are either or both of:

CUSTOM

Specifies the prior probabilities of the classes within the dataset via a set of class name and probability pairs (for example, `CUSTOM ('red'=0.5 'green'=0.3 'blue'=0.2)`). By default, the prior probability for a class is based on the fraction of observations in the dataset (or active subset of the dataset) that belong to that class.

COST

Specifies the cost matrix used to adjust the prior probabilities. Each entry specifies the cost of misclassifying an observation that actually belongs to class0 as class1 instead (for example, `COST (('green' 'red')=3 ('green' 'blue')=2)`). By default, the cost is 0 (zero) if the classes are the same, and 1 if they are not.

PROBABILITY

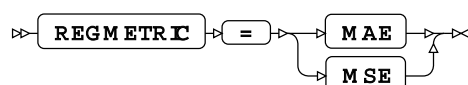
Specifies the method used by the procedure for estimating probabilities. Valid values are as follows:

NONE

Specifies that no probability estimation be performed. This is the default.

PLATT

Specifies that the Platt Scaling method is used for probability estimation. If this option is specified, the score results table will include probability estimates.

REGMETRIC

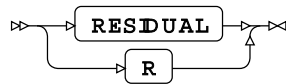
For SVM regression modelling (`TYPE=REG` in the `PROC SVM` statement), specifies the metric to use when selecting the model. Valid values are as follows:

MAE

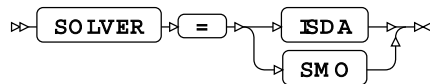
Specifies that the mean average error is used. This is the default.

MSE

Specifies that the mean squared error is used.

RESIDUAL

Specifies that residual values are included in the OutputStats table. This option is ignored unless `PROC SVM` specifies `TYPE=REG`.

SOLVER

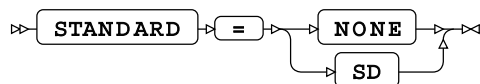
Specifies the algorithm used to solve the optimisation problem when constructing classifiers.

ISDA

Specifies the Iterative Single Data Algorithm (ISDA) is used to solve the optimization problem.

SMO

Specifies the Sequential Minimal Optimization (SMO) algorithm is used to solve the optimization problem. This is the default value.

STANDARD

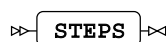
Specifies the method used to standardise the predictor variable values.

NONE

Specifies that no predictor scaling should be applied. This is the default.

SD

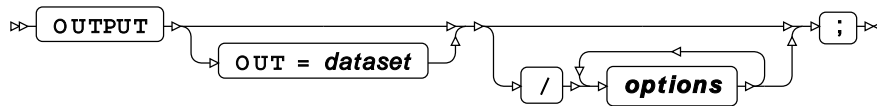
Specifies that the predictors should be centred and scaled by the respective means and standard deviations.

STEPS

Specifies that all evaluated models, including the intermediate models, are displayed. Otherwise, the only results that are displayed are those from the model that is finally selected.

OUTPUT

Creates a new dataset containing the input observations and the predicted values calculated by the SVM model.



All the variables in the original dataset are included in the output dataset, along with variables created by the `OUTPUT` statement. The new variables contain the predicted values calculated for each observation in the dataset.

OUT

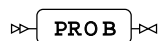
Specifies the name of the output dataset.

If `OUT` is not specified, the procedure creates the dataset as `DATAN` in the `WORK` library, where `n` is incremented for each output dataset.

Options

The following *options* are available:

PROB

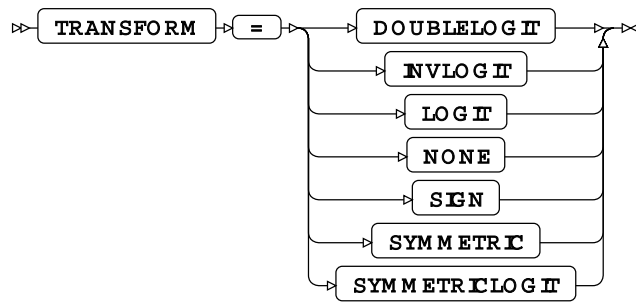


For classification modelling, specifies that posterior probability estimates for each class are included in the output dataset. This option is ignored if `PROC SVM` specifies `TYPE=REG`.

RESIDUAL



For regression modelling, specifies that residual values are included in the output dataset. This option is ignored unless `PROC SVM` specifies `TYPE=REG`.

TRANSFORM

For classification modelling, specifies that the score results are transformed by a transform function $f(x)$ before being output. This option is ignored if `PROC SVM` specifies `TYPE=REG`.

The following transform functions can be specified:

DOUBLELOGIT

$$f(x) = (1 + \exp(-2x))^{-1}$$

INVLOGIT

$$f(x) = \ln \frac{x}{1-x}$$

LOGIT

$$f(x) = \frac{1}{1 + \exp(-x)}$$

NONE

No transformation is applied. This is the default value.

SIGN

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

SYMMETRIC

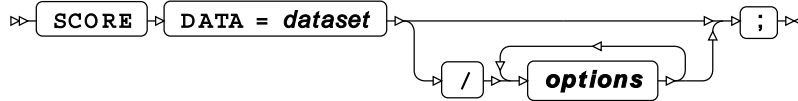
$$f(x) = 2x - 1$$

SYMMETRICLOGIT

$$f(x) = \frac{2}{1 + \exp(-x)} - 1$$

SCORE

Uses the current SVM model to score or classify the data in the specified dataset.



The **SCORE** statement takes the data in the specified dataset and scores it using the SVM model defined in **PROC SVM**. The score results are saved in a table which can be printed or saved in an output dataset.

The score results table includes all the data in the input dataset, including observations with missing values. But observations with missing values for predictor variables are not scored.

Multiple **SCORE** statements can be specified if required.

You can use the **PRINT** option to print the score results. You can also use **ODS OUTPUT** to save the score results in an output dataset. If the model has one or two predictor variables, you can also produce score plots for the mixture model and its components. Models with one predictor variable produce line plots, and models with two predictor variables produce contour plots.

DATA

Specifies the dataset to score. All the predictor variables specified in the **MODEL** statement must be present in the dataset. If a predictor variable has been specified as categorical using the **CLASS** statement, the categories in the dataset must match the categories specified in the **CLASS** statement.

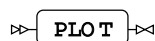
If **PROC GMM** includes a **BY** statement, the dataset to be scored must also contain all the variables mentioned in the **BY** statement, and must be sorted in the order of those variables.

The **DATA** option is mandatory.

Options

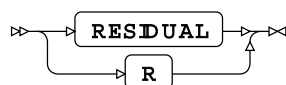
The following *options* are available:

PLOT



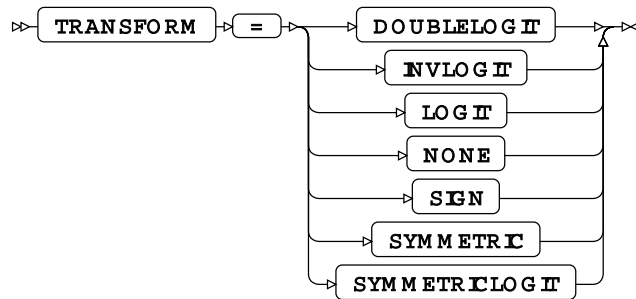
Specifies that score results plots are output. This option is ignored if the **PLOTS** option in the **PROC SVM** statement has not been specified.

RESIDUAL



For regression modelling, specifies that residual values are included in the score results table. This option is ignored unless PROC SVM specifies TYPE=REG.

TRANSFORM



For classification modelling, specifies that the score results are transformed by a transform function $f(x)$ before being output. This option is ignored if PROC SVM specifies TYPE=REG.

The following transform functions can be specified:

DOUBLELOGIT

$$f(x) = (1 + \exp(-2x))^{-1}$$

INVLOGIT

$$f(x) = \ln \frac{x}{1-x}$$

LOGIT

$$f(x) = \frac{1}{1 + \exp(-x)}$$

NONE

No transformation is applied. This is the default value.

SIGN

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

SYMMETRIC

$$f(x) = 2x - 1$$

SYMMETRICLOGIT

$$f(x) = \frac{2}{1 + \exp(-x)} - 1$$

WEIGHT

Specifies a variable in the input dataset giving the prior weight associated with each observation.

➤ **WEIGHT** ➤ *variable* ➤ ; ➤

SVM bibliography

These items are referenced in the SVM procedure section.

- [1] Boser, B., Guyon, I. & Vapnik, V., 1992. A training algorithm for optimal margin classifiers. In: D. Haussler, ed. *Proceedings of the Annual Conference on Computational Learning Theory*. Pittsburgh, Pennsylvania: ACM Press, pp. 144–152.
- [2] Guyon, I., Boser, B. & Vapnik, V., 1993. Automatic capacity tuning of very large VC-dimension classifiers. In: S. J. Hanson, J. D. Cowan & C. L. Giles, eds. *Advances in Neural Information Processing Systems 5*. s.l.:Morgan Kaufmann Publishers, pp. 147–155.
- [3] Cortes, C. & Vapnik, V., 1995. Support vector networks. *Machine Learning*, Issue 20, pp. 273–297.
- [4] Scholkopf, B., Burges, C. & Vapnik, V., 1995. Extracting support data for a given task. In: U. M. Fayyad & R. Uthurusamy, eds. *Proceedings, First International Conference on Knowledge Discovery and Data Mining*. Menlo Park: AAAI Press.
- [5] Scholkopf, B., Burges, C. & Vapnik, V., 1996. Incorporating invariances in support vector learning machines. In: C. von der Malsburg, W. von Seelen, J. C. Vorbruggen & B. Sendhoff, eds. *Artificial Neural Networks ICANN'96*. Berlin: Springer, pp. 47–52.
- [6] Vapnik, V., Golowich, S. & Smola, A., 1997. Support Vector Method for Function Approximation, Regression Estimation and Signal Processing. In: M. C. Mozer, M. I. Jordan & T. Petsche, eds. *Advances in Neural Information Processing Systems 9*. Cambridge, Massachusetts: MIT Press, pp. 281–287.
- [7] Vapnik, V. & Lerner, A., 1963. Pattern Recognition using Generalized Portrait Method. *Automation and Remote Control*, Volume 24, pp. 774–780.
- [8] Aizerman, M. A., Braverman, E. M. & Rozonoer, L. I., 1964. Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Automation and Remote Control*, Issue 25, pp. 821–837.
- [9] Dietterich, T. G. & Bakiri, G., 1995. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, Issue 2, pp. 263–286.
- [10] Kecman, V., Huang, T.-M. & Vogt, M., 2005. Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance. In: L. Wang, ed. *Support Vector Machines: Theory and Applications*. Berlin: Springer-Verlag, pp. 255–274.
- [11] Platt, J. C., 1998. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*, s.l.: Microsoft Research.

- [12] Platt, J. C., 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in large margin classifiers*, 10(3), pp. 61–74.
- [13] Lin, H.-T., Lin, C.-J. & Weng, R. C., 2007. A Note on Platt's Probabilistic Outputs for Support Vector Machines. *Machine Learning*, 68(3), pp. 267–276.
- [14] Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), pp. 179–188.

WPS Operational Research

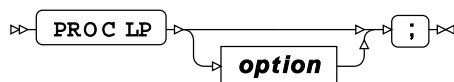
Operational research procedures

LP procedure

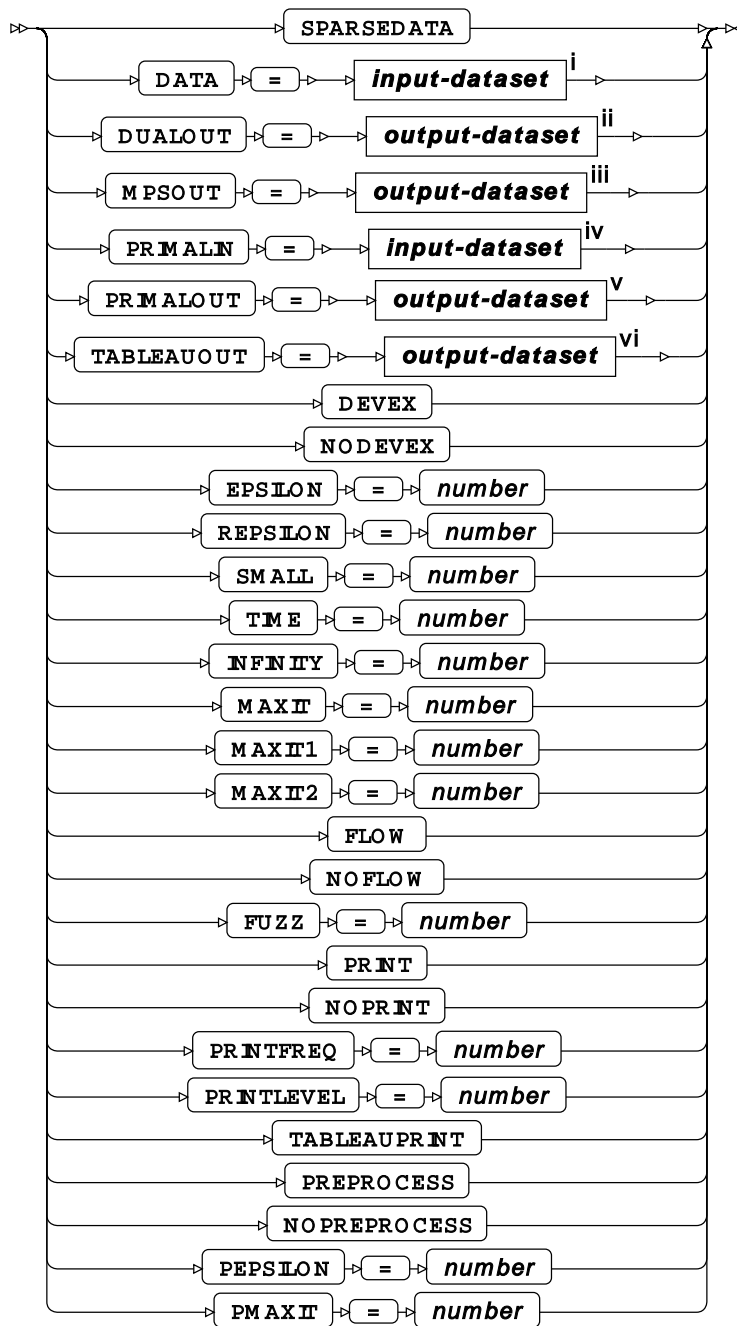
Supported statements

- *PROC LP* [↗](#) (page 3389)
- *ATTRIB* [↗](#) (page 3391)
- *COEF* [↗](#) (page 3391)
- *COL* [↗](#) (page 3391)
- *FORMAT* [↗](#) (page 3391)
- *INFORMAT* [↗](#) (page 3391)
- *LABEL* [↗](#) (page 3392)
- *RANGE* [↗](#) (page 3392)
- *RHSSEN* [↗](#) (page 3392)
- *ROW* [↗](#) (page 3392)
- *TYPE* [↗](#) (page 3392)
- *VAR* [↗](#) (page 3392)
- *WHERE* [↗](#) (page 3393)

PROC LP



option



ⁱ See [Input dataset](#) (page 16).

ⁱⁱ See [Output dataset](#) (page 16).

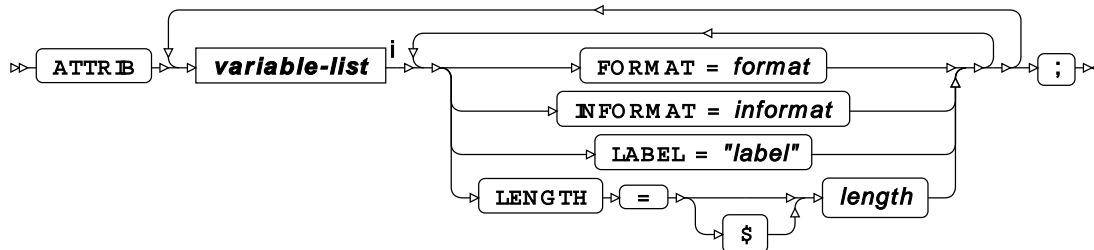
ⁱⁱⁱ See [Output dataset](#) (page 16).

^{iv} See [Input dataset](#) (page 16).

^v See [Output dataset](#) (page 16).

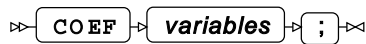
^{vi} See *Output dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

COEF



COL



FORMAT



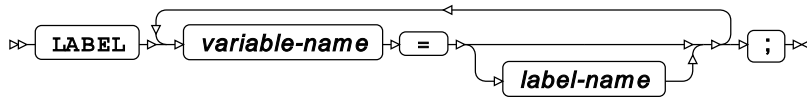
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



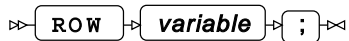
RANGE



RHSSEN



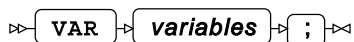
ROW



TYPE



VAR



WHERE



WPS Quality Control

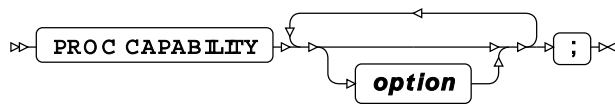
Quality control procedures

CAPABILITY Procedure

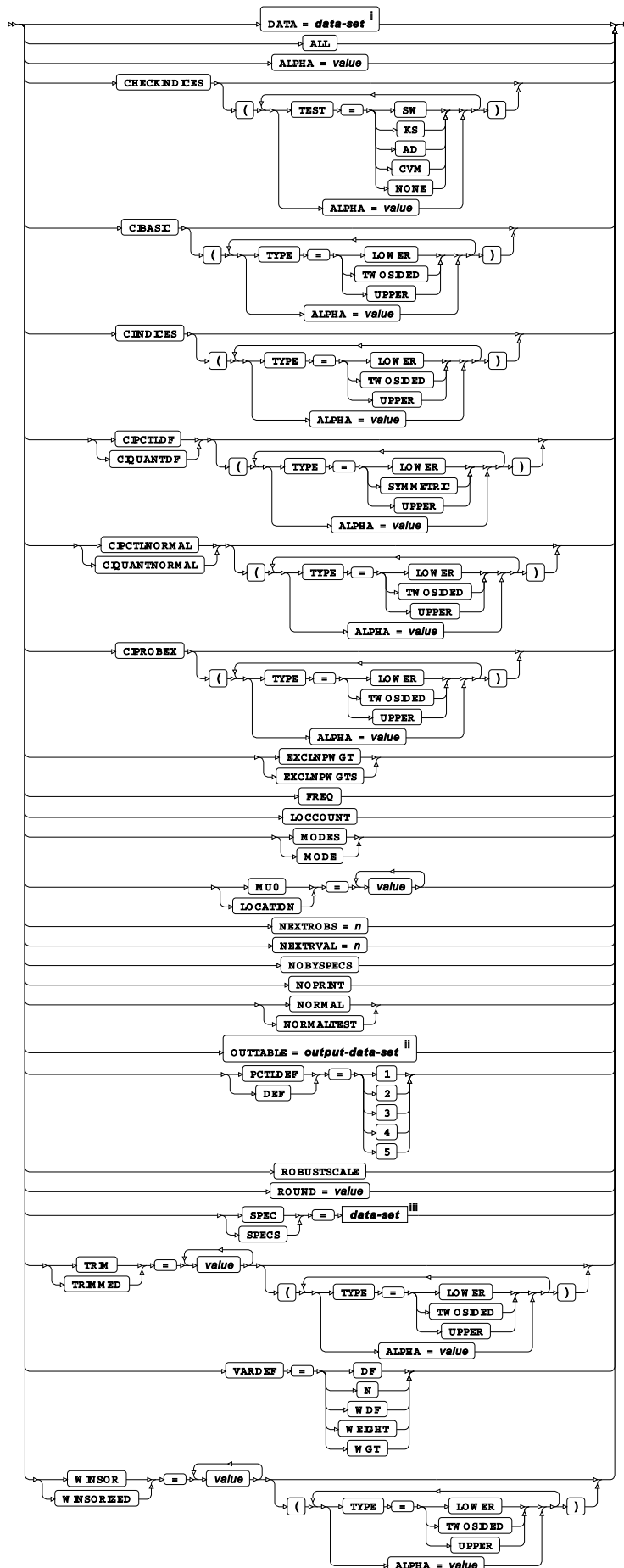
Supported statements

- *PROC CAPABILITY* [↗](#) (page 3395)
- *ATTRIB* [↗](#) (page 3397)
- *BY* [↗](#) (page 3397)
- *CDFPLOT* [↗](#) (page 3398)
- *CLASS* [↗](#) (page 3404)
- *COMPHISTOGRAM* [↗](#) (page 3405)
- *FORMAT* [↗](#) (page 3407)
- *FREQ* [↗](#) (page 3407)
- *HISTOGRAM* [↗](#) (page 3408)
- *ID* [↗](#) (page 3418)
- *INFORMAT* [↗](#) (page 3418)
- *LABEL* [↗](#) (page 3418)
- *OUTPUT* [↗](#) (page 3418)
- *PPLOT* [↗](#) (page 3422)
- *PROBPLOT* [↗](#) (page 3428)
- *QQPLOT* [↗](#) (page 3434)
- *SPEC* [↗](#) (page 3440)
- *VAR* [↗](#) (page 3440)
- *WEIGHT* [↗](#) (page 3440)
- *WHERE* [↗](#) (page 3440)

PROC CAPABILITY

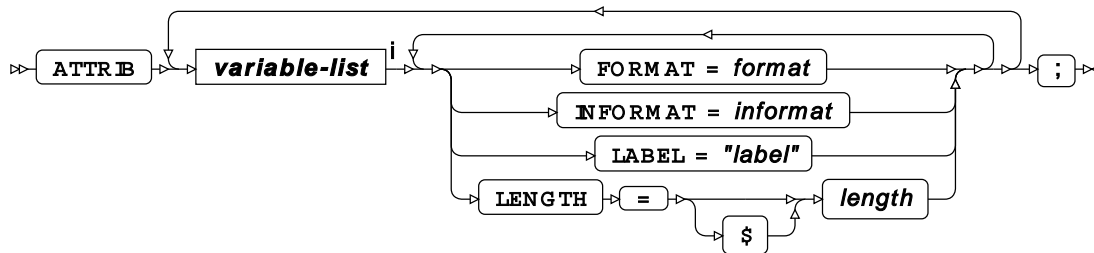


option



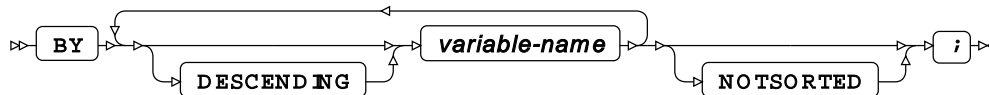
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Input dataset* [↗](#) (page 16).

ATTRIB

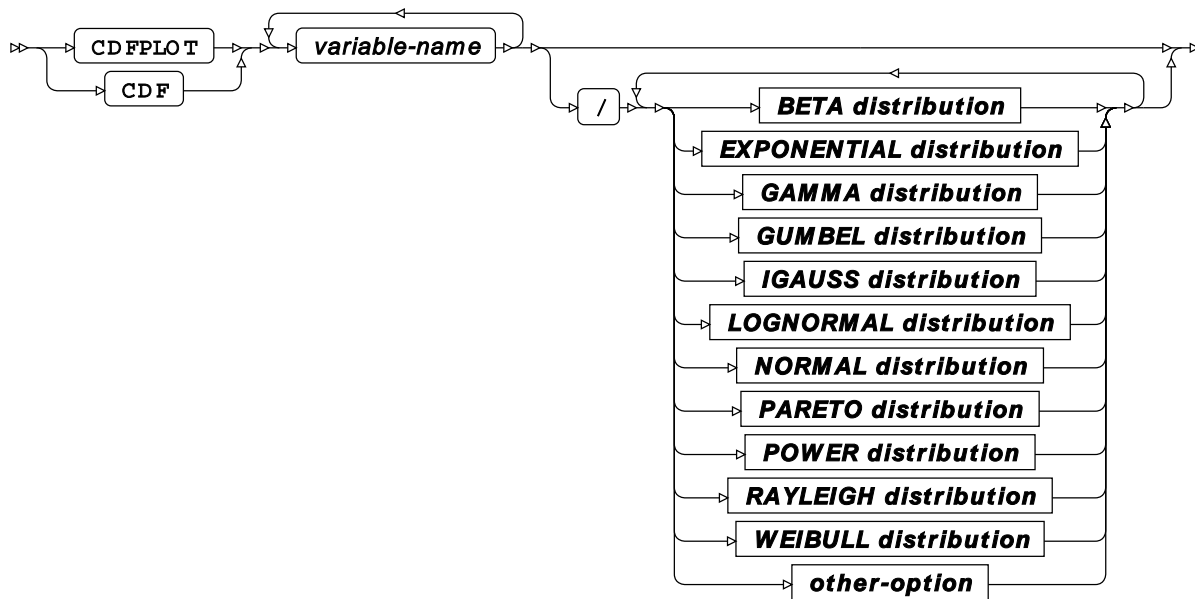


- ⁱ See *Variable Lists* [↗](#) (page 32).

BY

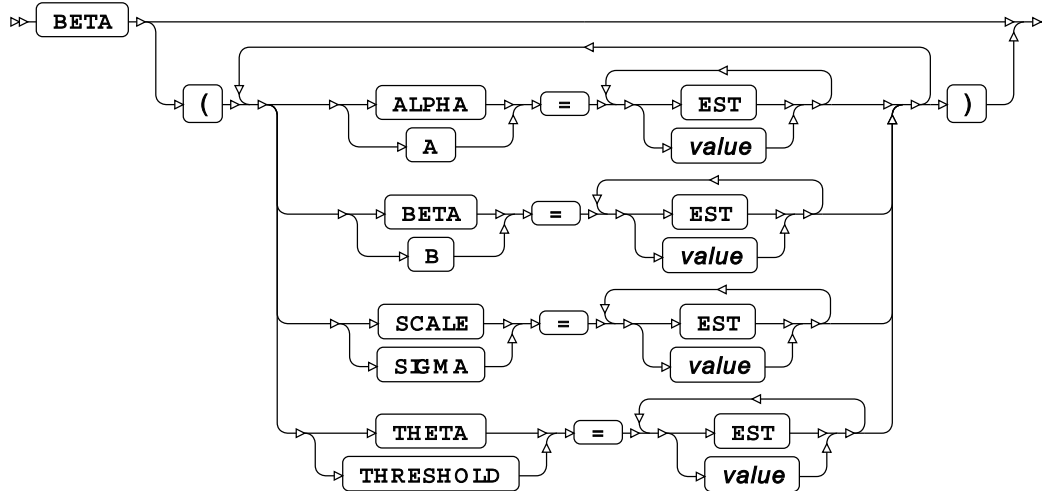


CDFPLOT

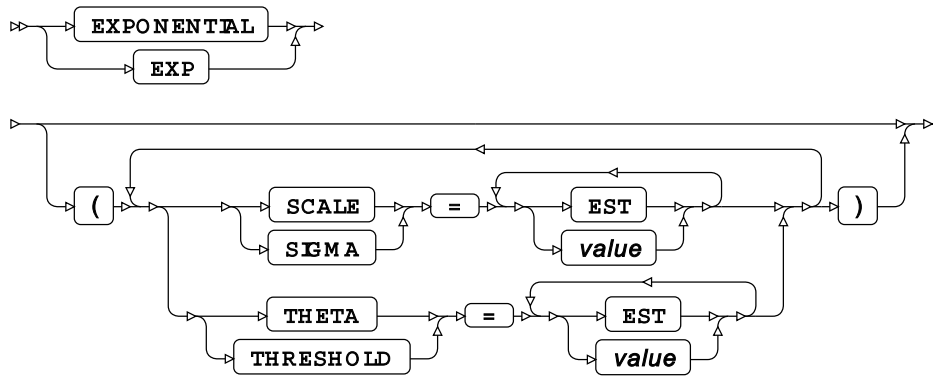


;

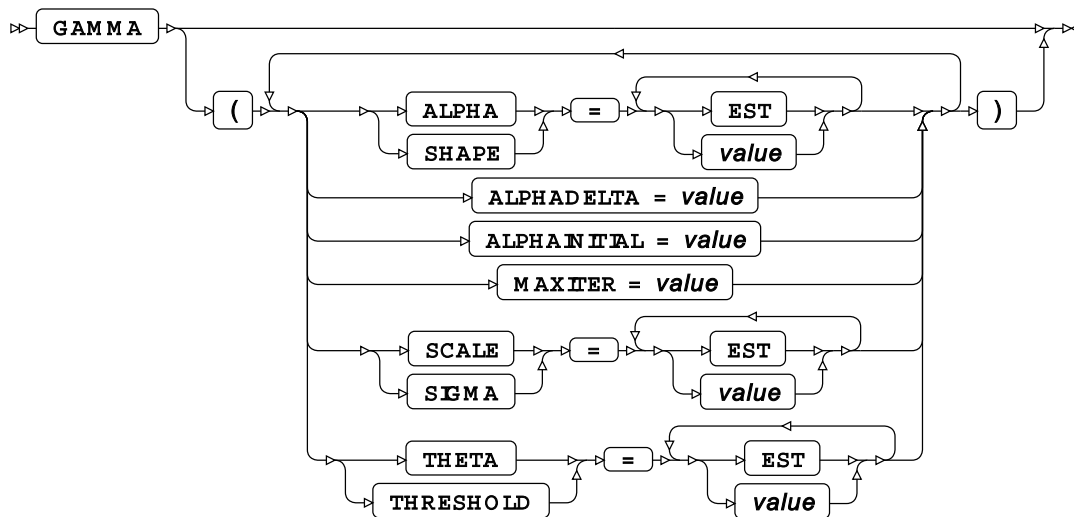
BETA distribution



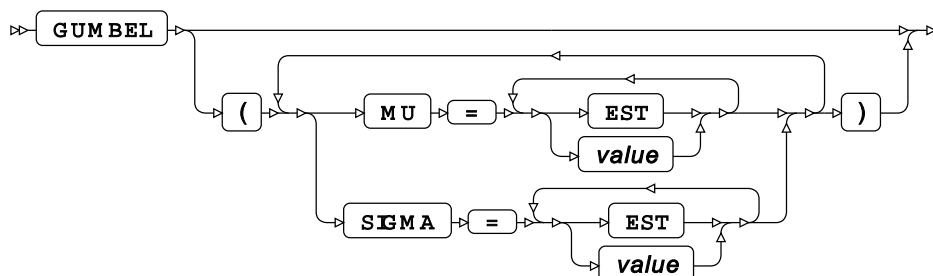
EXPONENTIAL distribution



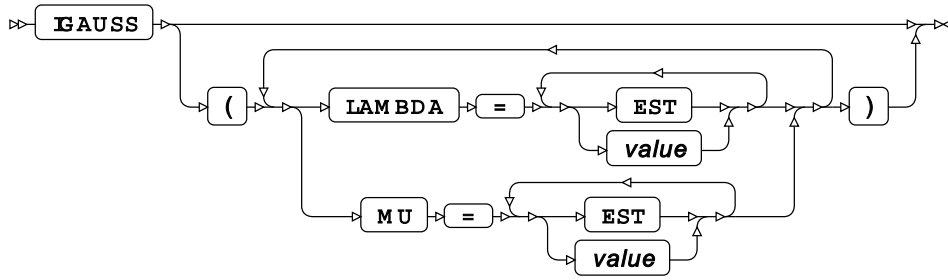
GAMMA distribution



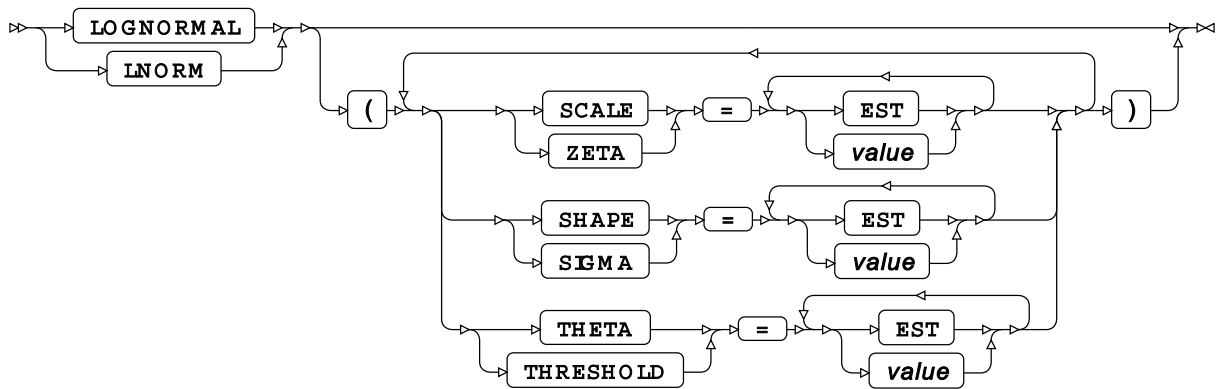
GUMBEL distribution



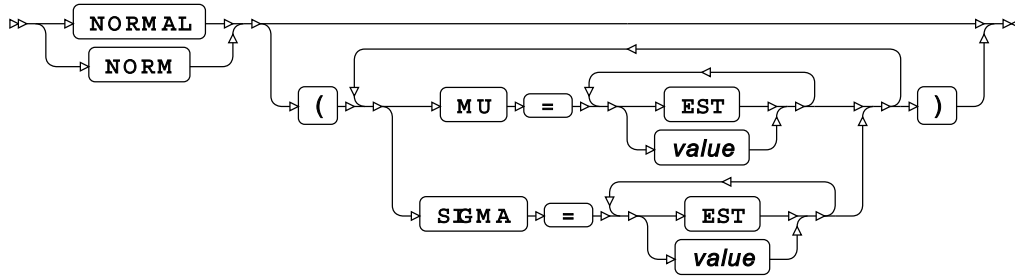
IGAUSS distribution



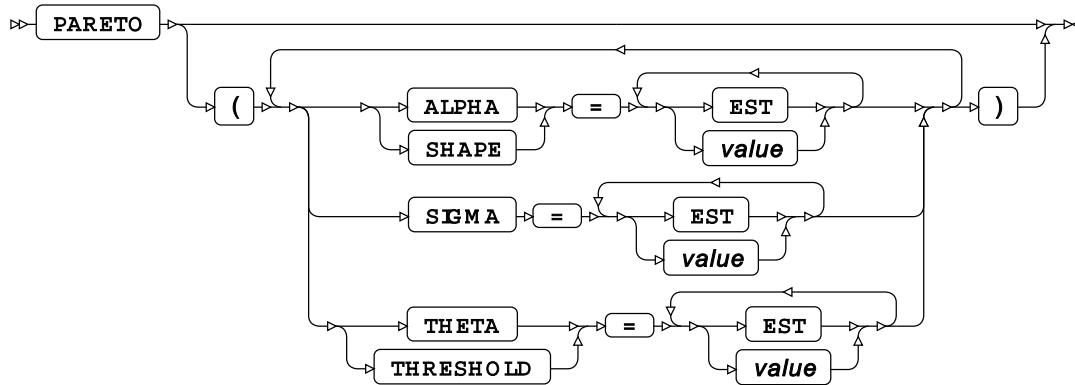
LOGNORMAL distribution



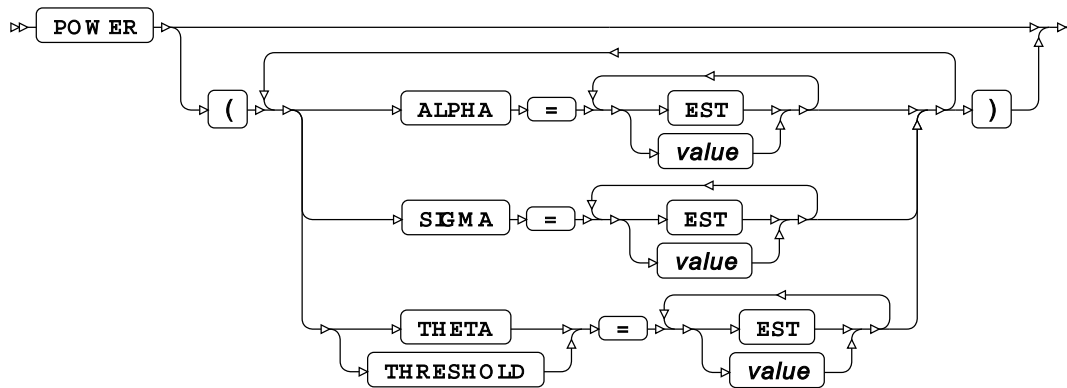
NORMAL distribution



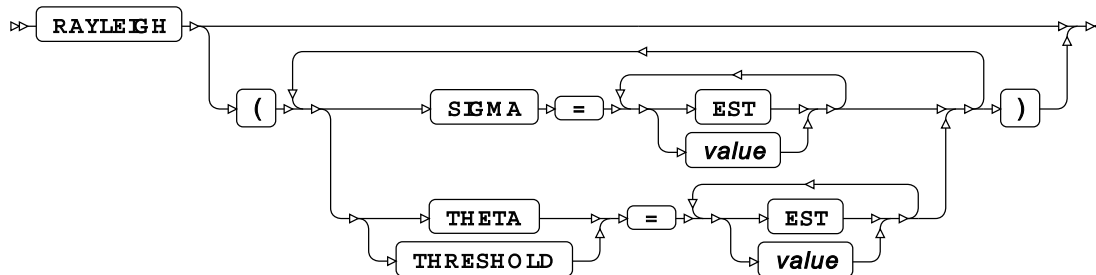
PARETO distribution



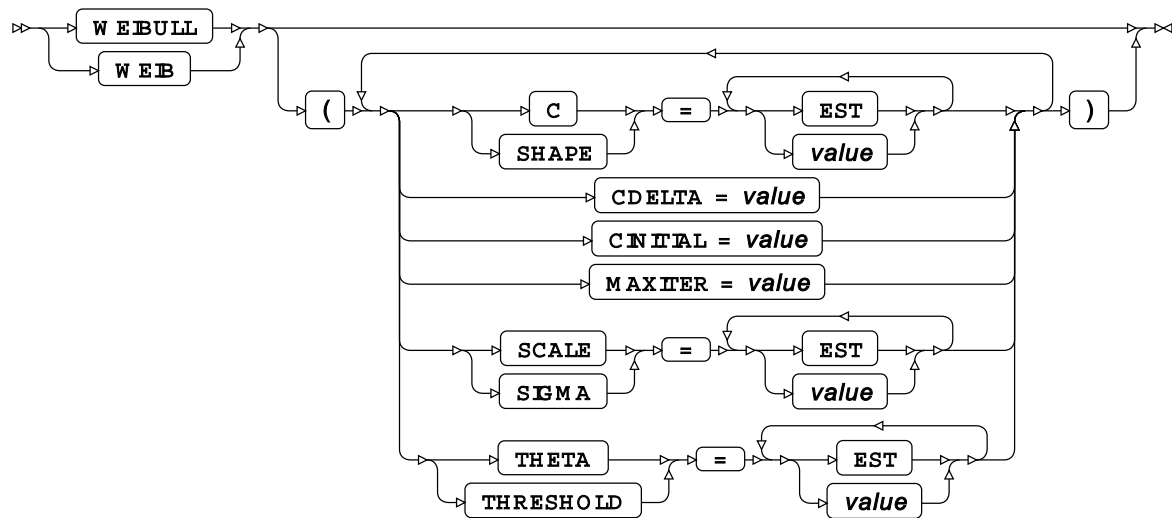
POWER distribution



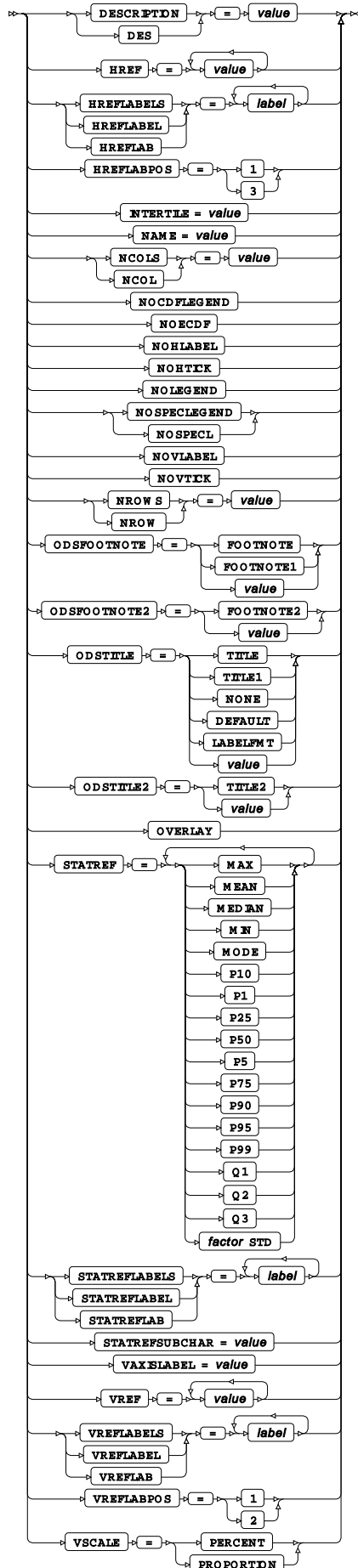
RAYLEIGH distribution



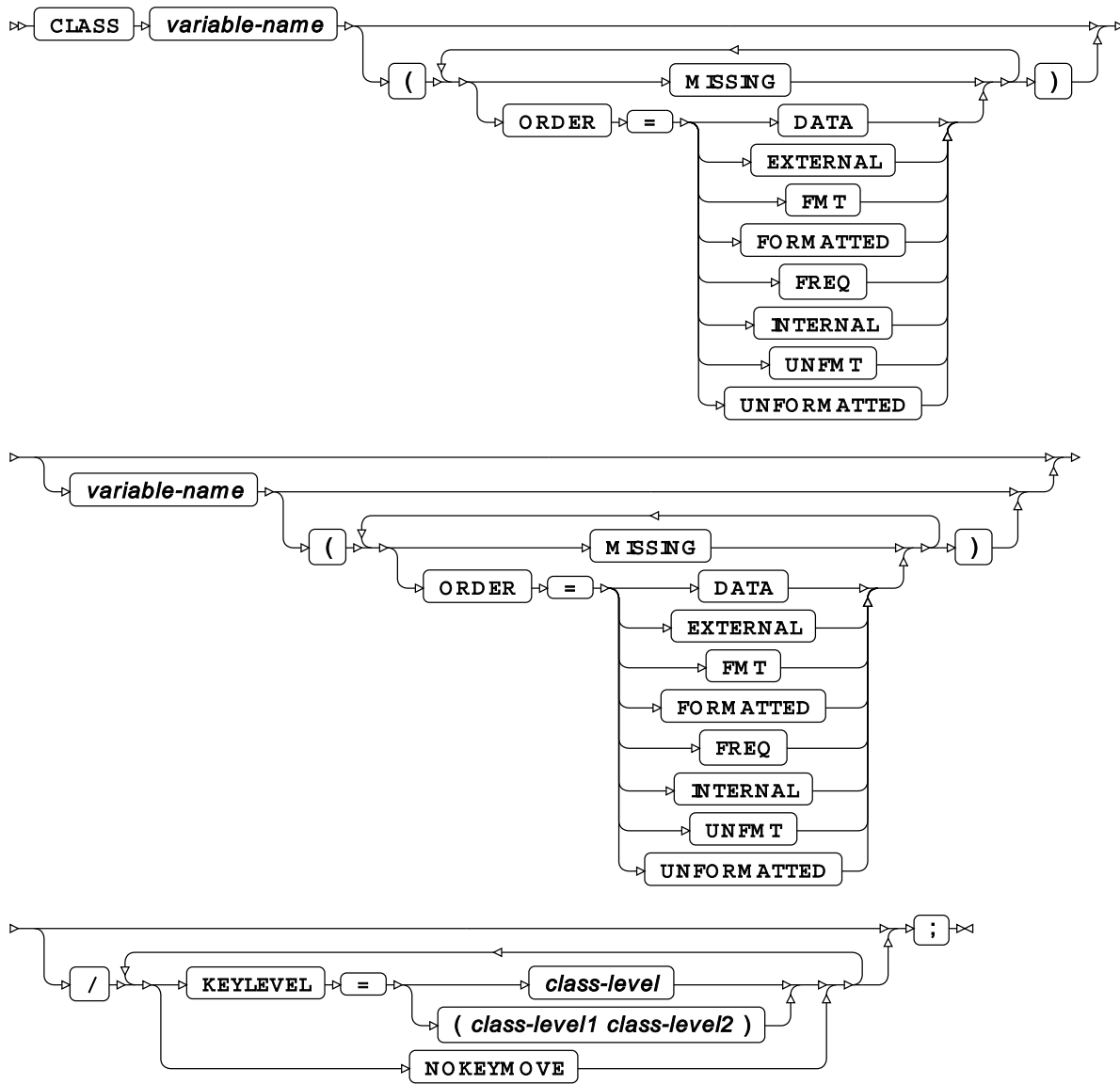
WEIBULL distribution



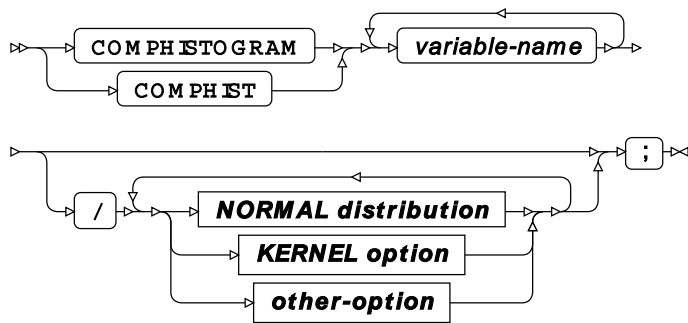
other-option



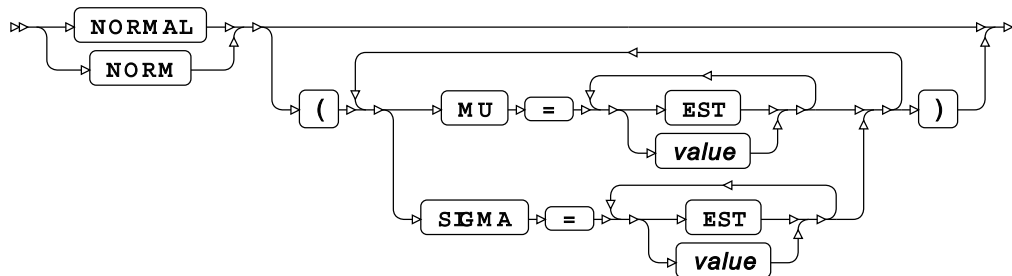
CLASS



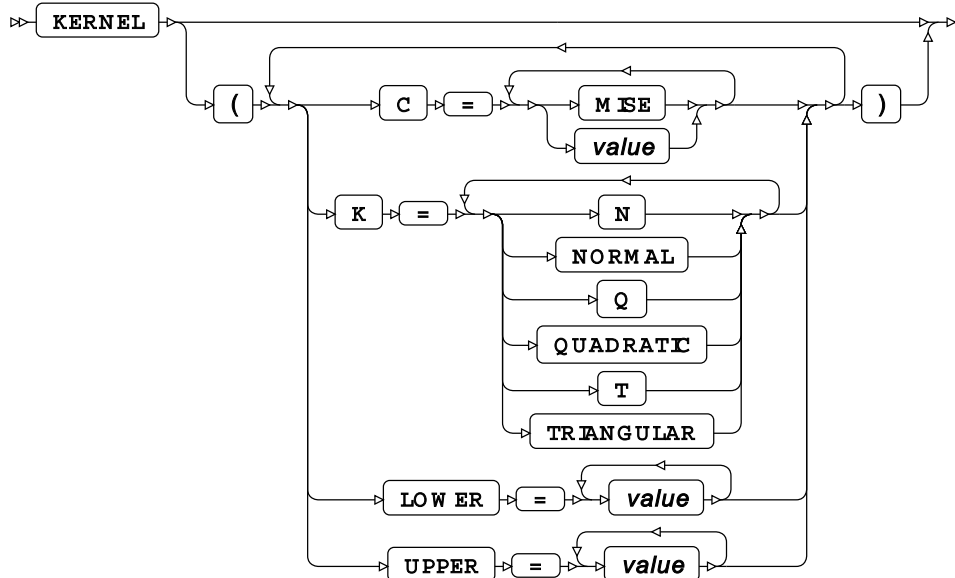
COMPHISTOGRAM



NORMAL distribution



KERNEL option

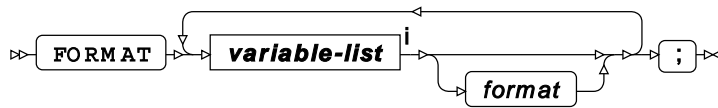


```

graph TD
    ODS_OUTPUT[ODS OUTPUT =] --> BARLABEL[BARLABEL =]
    BARLABEL --> COUNT[COUNT]
    BARLABEL --> PERCENT[PERCENT]
    BARLABEL --> PROPORTDN[PROPORTDN]
    ODS_OUTPUT --> CLASS[CLASS =]
    CLASS --> variable[variable]
    CLASS --> var_pair["( variable1 variable2 )"]
    ODS_OUTPUT --> CLASSKEY[CLASSKEY =]
    CLASSKEY --> value1[value]
    CLASSKEY --> var_pair2["( value1 value2 )"]
    ODS_OUTPUT --> DESCRPTDN[DESCRPTDN =]
    DESCRPTDN --> DES[DES]
    DESCRPTDN --> value2[value]
    ODS_OUTPUT --> ENDPOINTS[ENDPOINTS =]
    ENDPOINTS --> value3[value]
    ENDPOINTS --> TO[TO]
    ENDPOINTS --> value4[value]
    ENDPOINTS --> BY[BY]
    ENDPOINTS --> value5[value]
    ENDPOINTS --> KEY[KEY]
    ENDPOINTS --> UNIFORM[UNIFORM]
    ODS_OUTPUT --> HREF[HREF =]
    HREF --> value6[value]
    ODS_OUTPUT --> HREFLABELS[HREFLABELS =]
    HREFLABELS --> label[label]
    ODS_OUTPUT --> HREFLAB[HREFLAB]
    HREFLAB --> 1[1]
    HREFLAB --> 3[3]
    ODS_OUTPUT --> HREFLABPOS[HREFLABPOS =]
    HREFLABPOS --> 1_2[1]
    HREFLABPOS --> 3_2[3]
    ODS_OUTPUT --> INTERTLE[INTERTLE = value]
    ODS_OUTPUT --> MAXNBN[MAKNBN = value]
    ODS_OUTPUT --> MAXSEWAS[MAXSEWAS = value]
    ODS_OUTPUT --> MDPPOINTS[MDPPOINTS =]
    MDPPOINTS --> value7[value]
    MDPPOINTS --> TO2[TO]
    MDPPOINTS --> value8[value]
    MDPPOINTS --> BY2[BY]
    MDPPOINTS --> value9[value]
    MDPPOINTS --> KEY2[KEY]
    MDPPOINTS --> UNIFORM2[UNIFORM]
    ODS_OUTPUT --> MDSMSG1[MDSMSG1]
    ODS_OUTPUT --> MDSMSG2[MDSMSG2]
    ODS_OUTPUT --> NAME[NAME = value]
    ODS_OUTPUT --> NCOLS[NCOLS =]
    NCOLS --> value10[value]
    ODS_OUTPUT --> NCOL[NCOL]
    ODS_OUTPUT --> NENDPOINTS[NENDPOINTS = n]
    ODS_OUTPUT --> NM_DPPOINTS[NM DPPOINTS = n]
    ODS_OUTPUT --> NOBARS[NOBARS]
    ODS_OUTPUT --> NOHLABEL[NOHLABEL]
    ODS_OUTPUT --> NOHTYCK[NOHTYCK]
    ODS_OUTPUT --> NOKEYMOVE[NOKEYMOVE]
    ODS_OUTPUT --> NOPLOT[NOPLOT]
    ODS_OUTPUT --> NOCHART[NOCHART]
    ODS_OUTPUT --> NOVLABEL[NOVLABEL]
    ODS_OUTPUT --> NOVTYCK[NOVTYCK]
    ODS_OUTPUT --> NROWS[NROWS =]
    NROWS --> value11[value]
    ODS_OUTPUT --> NROW[NROW]
    ODS_OUTPUT --> ODSFOOTNOTE[ODSFOOTNOTE =]
    ODSFOOTNOTE --> FOOTNOTE[FOOTNOTE]
    ODSFOOTNOTE --> FOOTNOTE1[FOOTNOTE1]
    ODSFOOTNOTE --> value12[value]
    ODS_OUTPUT --> ODSFOOTNOTE2[ODSFOOTNOTE2 =]
    ODSFOOTNOTE2 --> FOOTNOTE2[FOOTNOTE2]
    ODSFOOTNOTE2 --> value13[value]
    ODS_OUTPUT --> ODSTYLE[ODSTYLE =]
    ODSTYLE --> TITLE[TITLE]
    ODSTYLE --> TITLE1[TITLE1]
    ODSTYLE --> NONE[NONE]
    ODSTYLE --> DEFAULT[DEFAULT]
    ODSTYLE --> LABELPMT[LABELPMT]
    ODSTYLE --> value14[value]
    ODS_OUTPUT --> ODSTYLE2[ODSTYLE2 =]
    ODSTYLE2 --> TITLE2[TITLE2]
    ODSTYLE2 --> value15[value]
    ODS_OUTPUT --> ORDER1[ORDER1 =]
    ORDER1 --> DATA[DATA]
    ORDER1 --> EXTERNAL[EXTERNAL]
    ORDER1 --> FMT[FMT]
    ORDER1 --> FORMATTED[FORMATTED]
    ORDER1 --> FREQ[FREQ]
    ORDER1 --> INTERNAL[INTERNAL]
    ORDER1 --> UNFMT[UNFMT]
    ORDER1 --> UNFORMATTED[UNFORMATTED]
    ODS_OUTPUT --> ORDER2[ORDER2 =]
    ORDER2 --> DATA2[DATA]
    ORDER2 --> EXTERNAL2[EXTERNAL]
    ORDER2 --> FMT2[FMT]
    ORDER2 --> FORMATTED2[FORMATTED]
    ORDER2 --> FREQ2[FREQ]
    ORDER2 --> INTERNAL2[INTERNAL]
    ORDER2 --> UNFMT2[UNFMT]
    ORDER2 --> UNFORMATTED2[UNFORMATTED]
    ODS_OUTPUT --> OUTHETOGRAM[OUTHETOGRAM]
    OUTHETOGRAM --> output_data_set1[output-data-setI]
    ODS_OUTPUT --> OUTHSET[OUTHSET]
    OUTHSET --> output_data_set2[output-data-setII]
    ODS_OUTPUT --> OUTKENEL[OUTKENEL]
    OUTKENEL --> output_data_set3[output-data-setII]
    ODS_OUTPUT --> OUTKINODE[OUTKINODE]
    OUTKINODE --> value16[value]
    ODS_OUTPUT --> VAXELABEL[VAXELABEL = value]
    ODS_OUTPUT --> VREF[VREF =]
    VREF --> value17[value]
    ODS_OUTPUT --> VREFLABELS[VREFLABELS =]
    VREFLABELS --> label2[label]
    ODS_OUTPUT --> VREFLAB[VREFLAB]
    VREFLAB --> 1_3[1]
    VREFLAB --> 3_3[3]
    ODS_OUTPUT --> VREFLABPOS[VREFLABPOS =]
    VREFLABPOS --> 1_4[1]
    VREFLABPOS --> 3_4[3]
    ODS_OUTPUT --> VSCALE[VSCALE =]
    VSCALE --> COUNT2[COUNT]
    VSCALE --> PERCENT2[PERCENT]
    VSCALE --> PROPORTDN2[PROPORTDN]
  
```

- ⁱ See *Output dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).

FORMAT

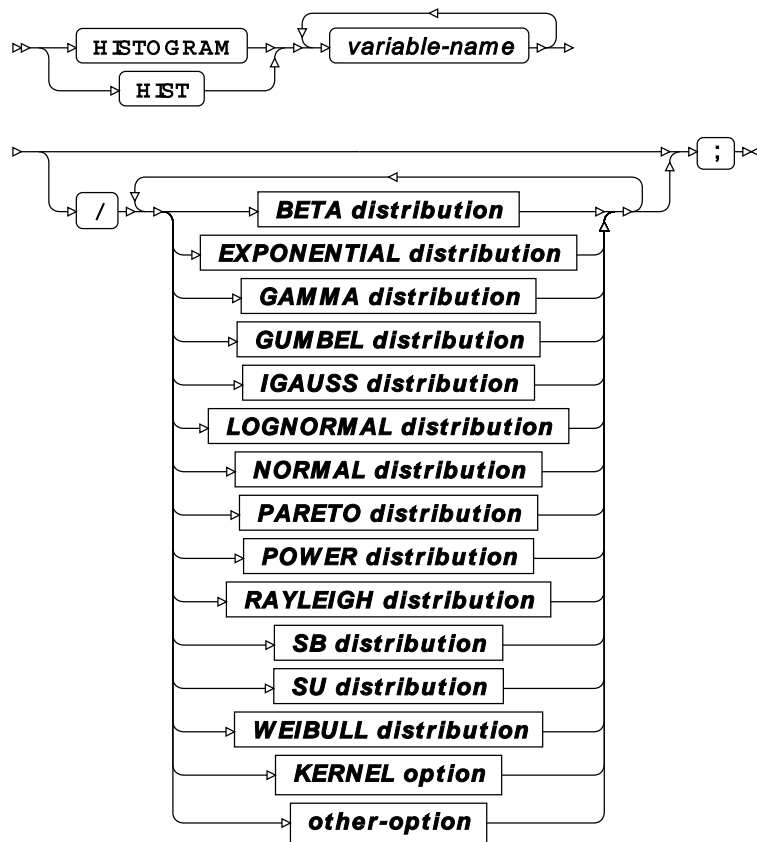


- ⁱ See *Variable Lists* [↗](#) (page 32).

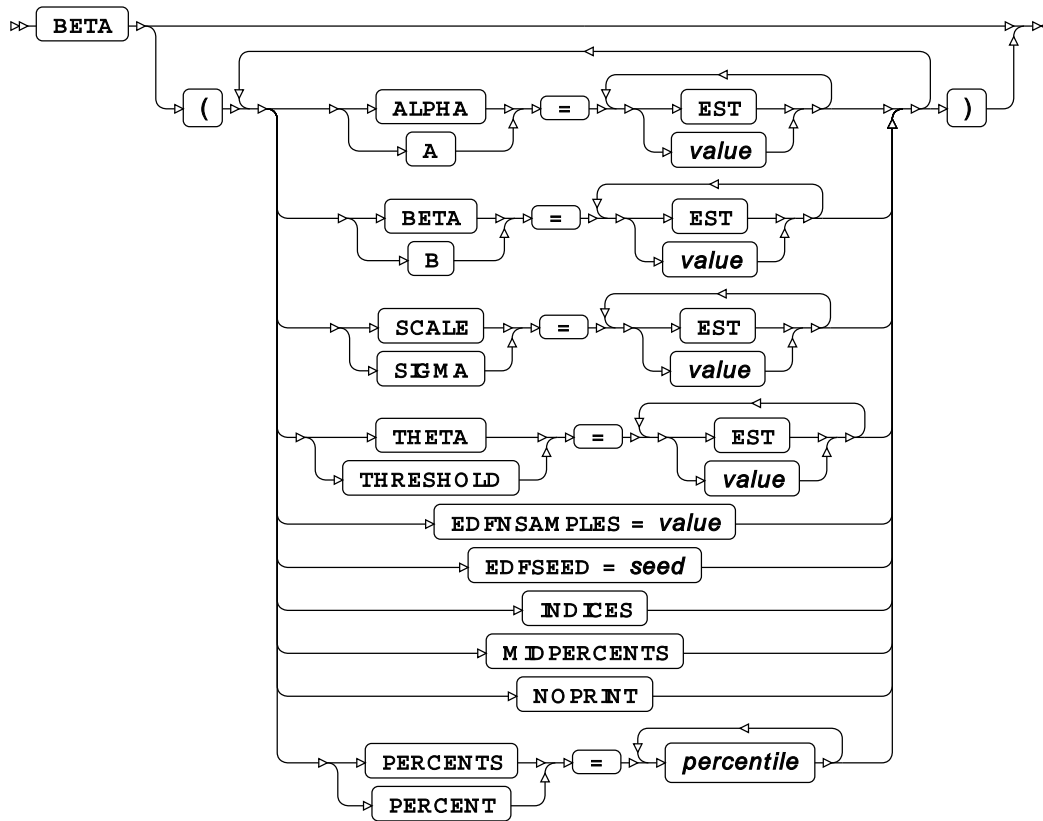
FREQ



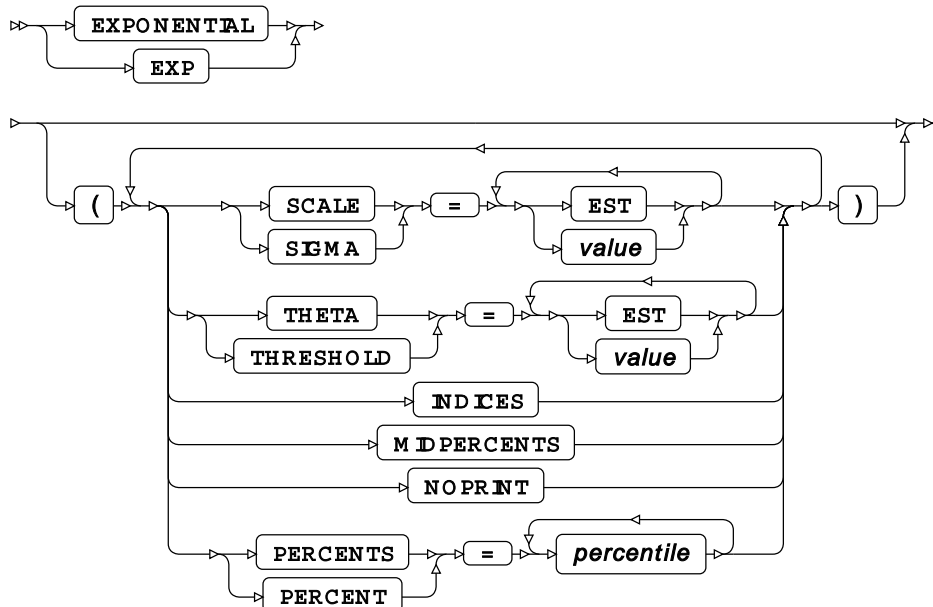
HISTOGRAM



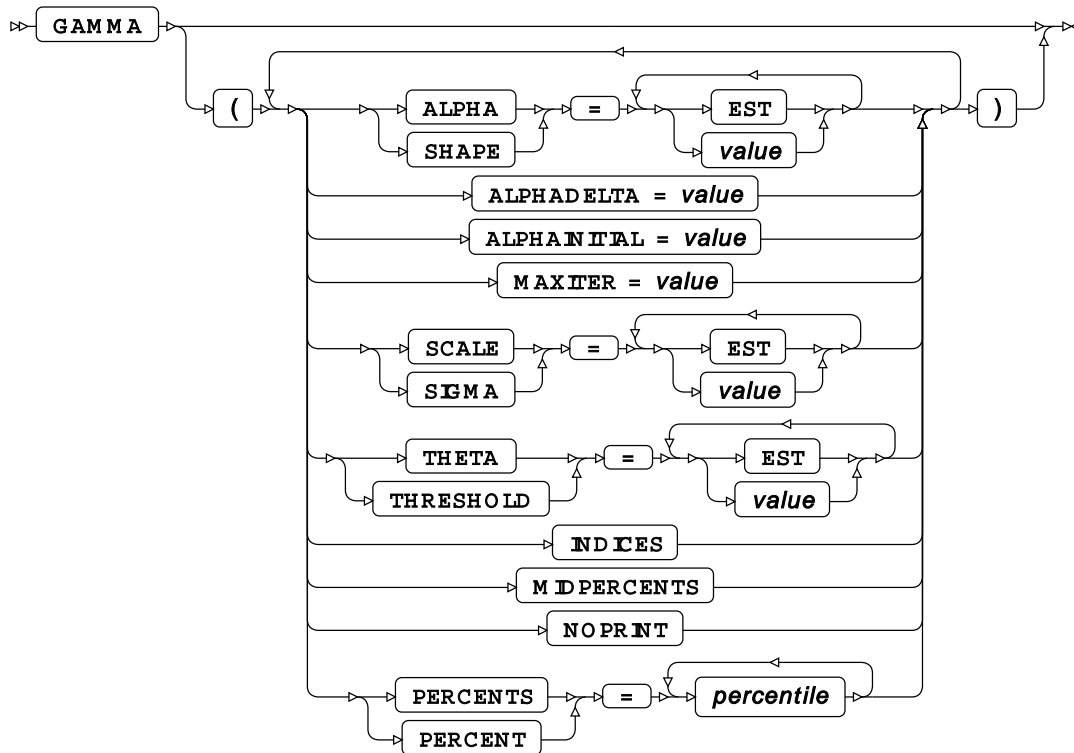
BETA distribution



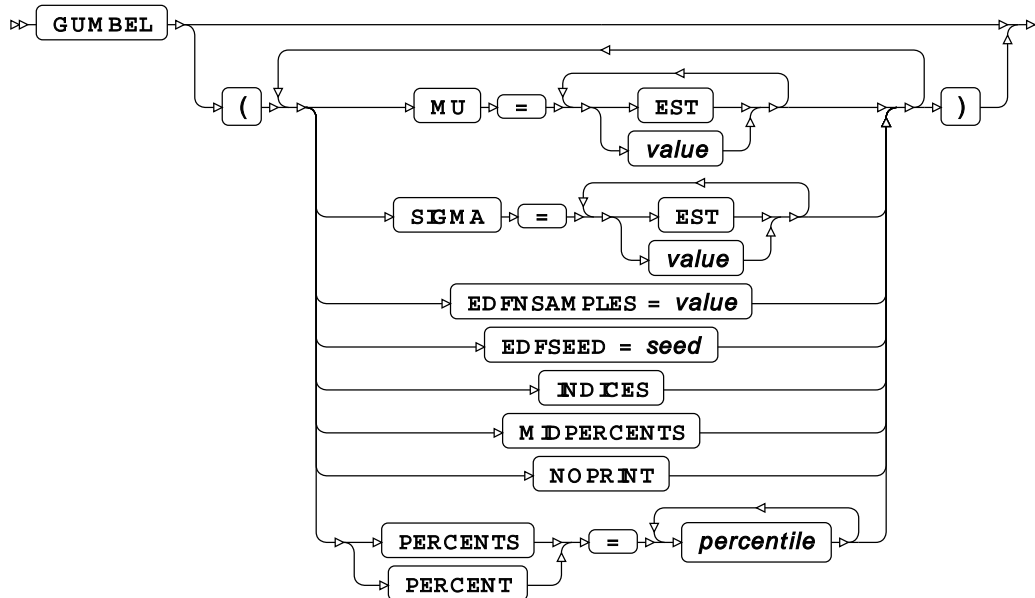
EXPONENTIAL distribution



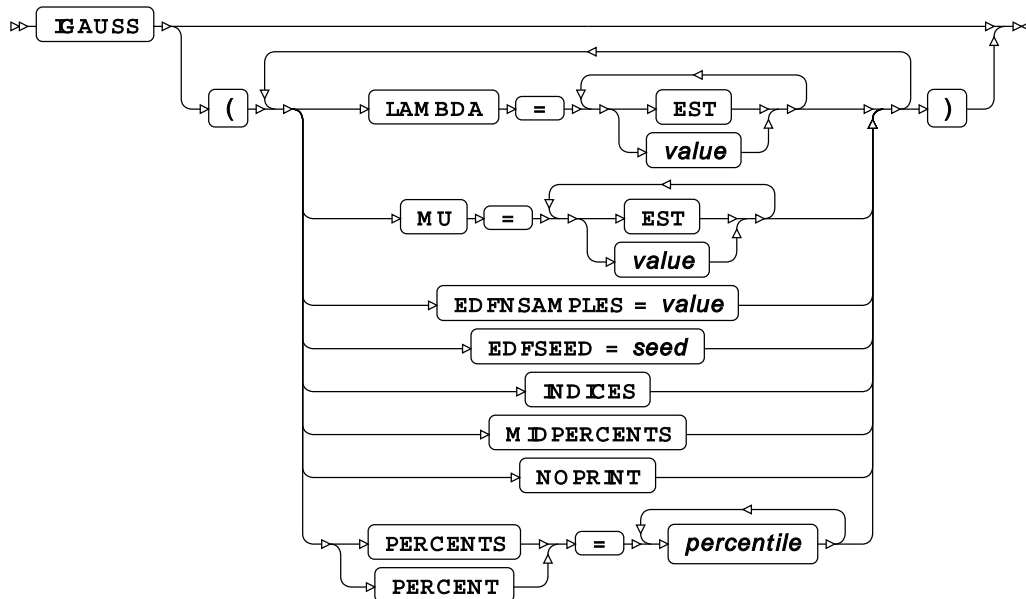
GAMMA distribution



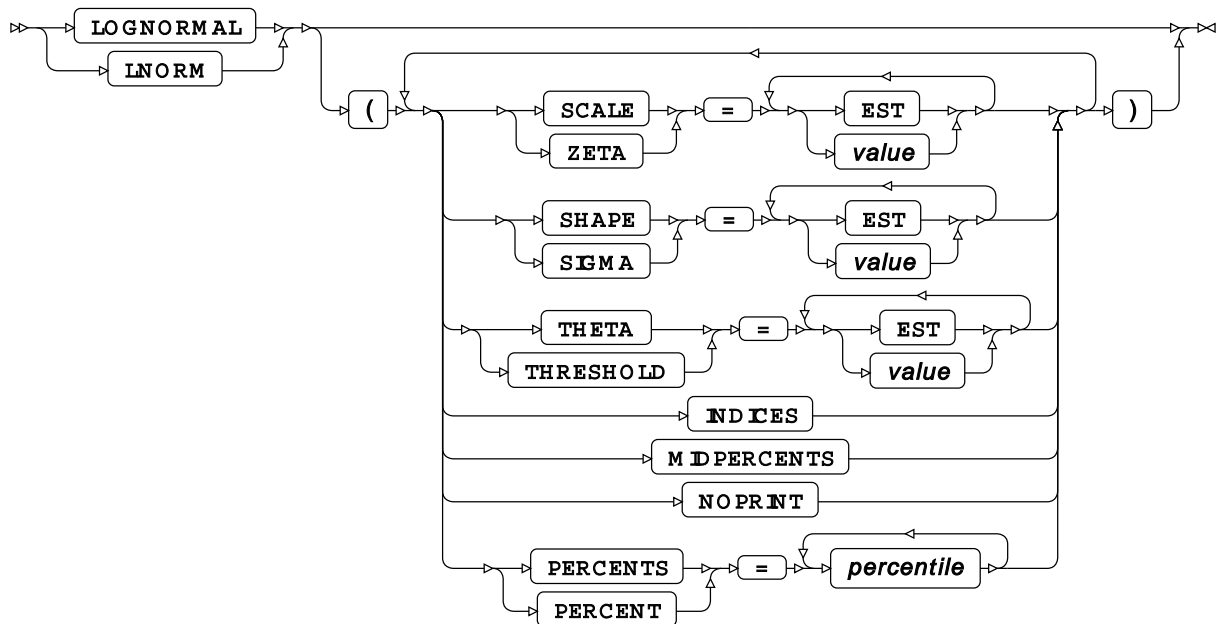
GUMBEL distribution



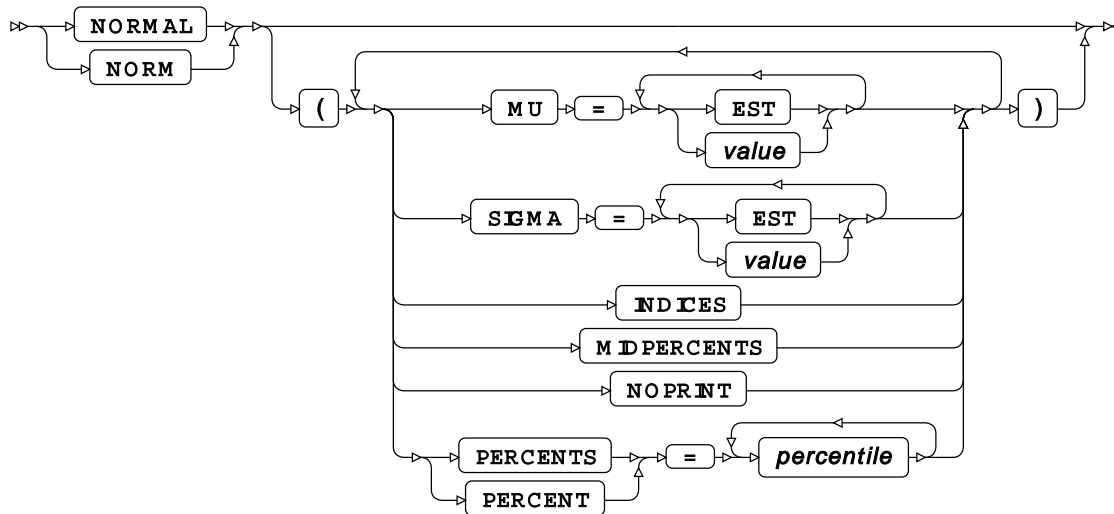
IGAUSS distribution



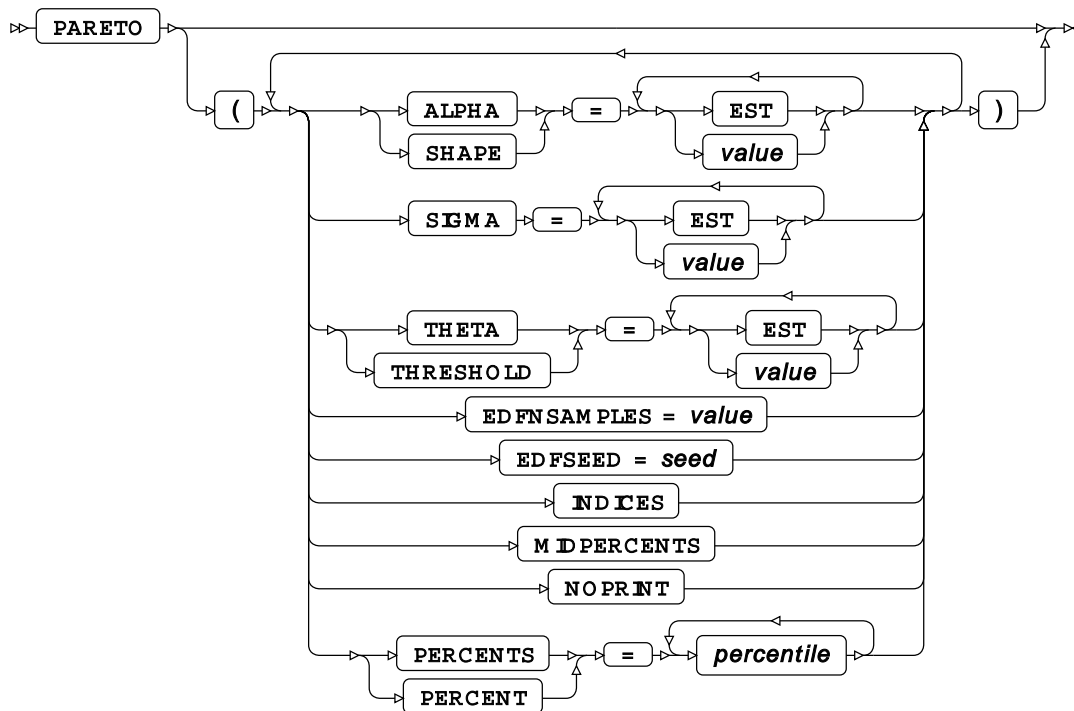
LOGNORMAL distribution



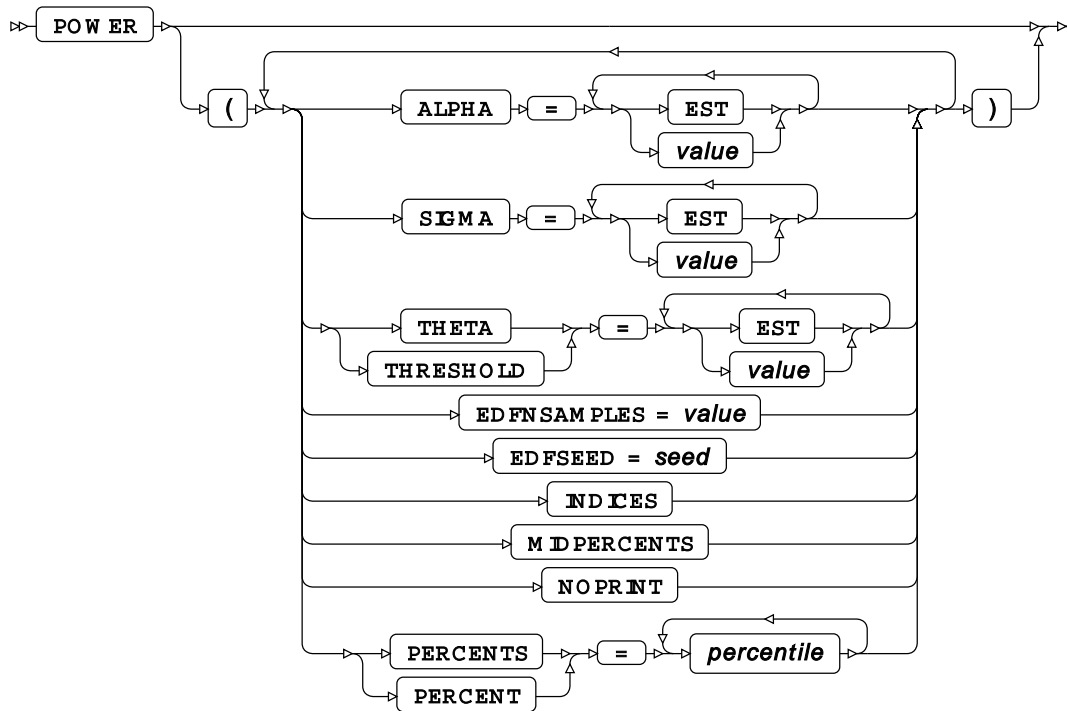
NORMAL distribution



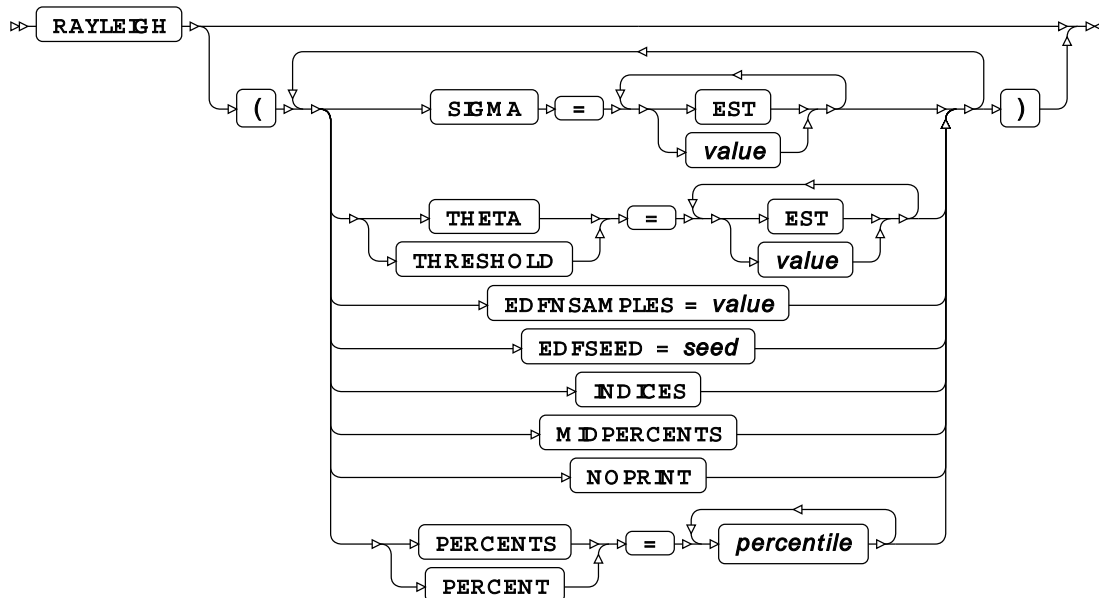
PARETO distribution



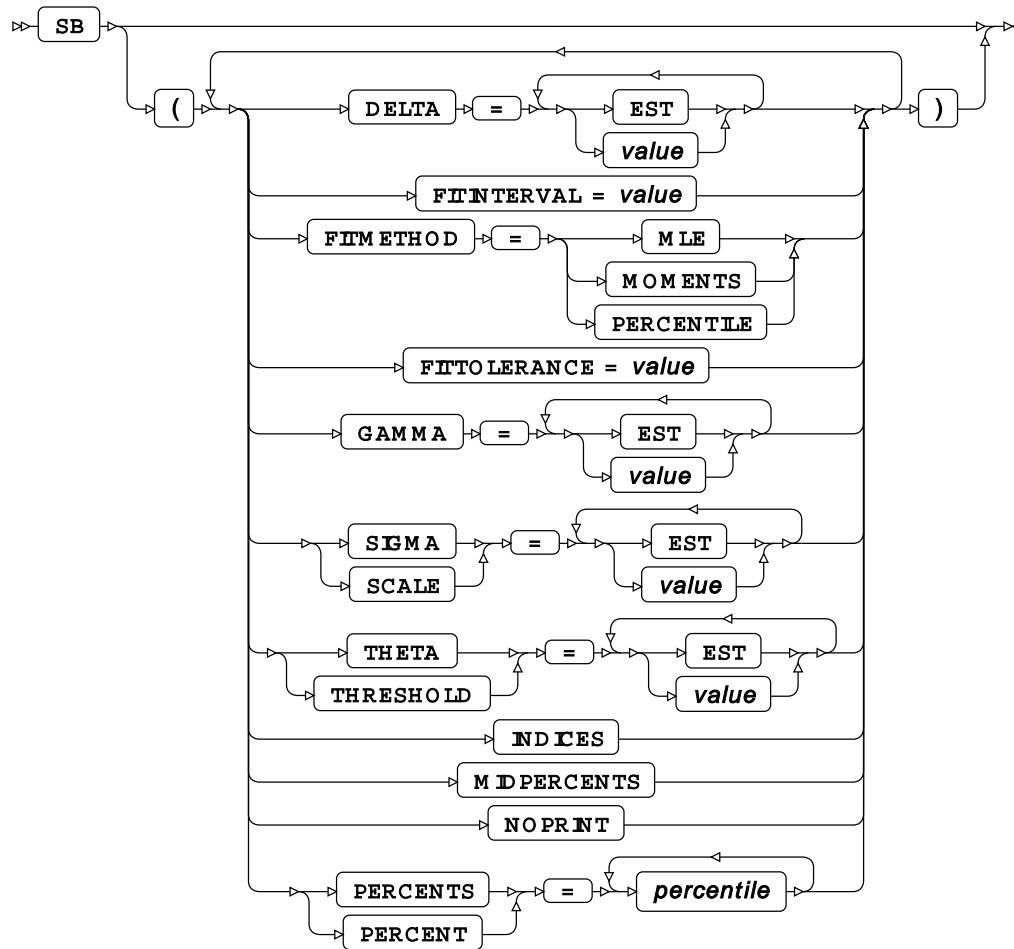
POWER distribution



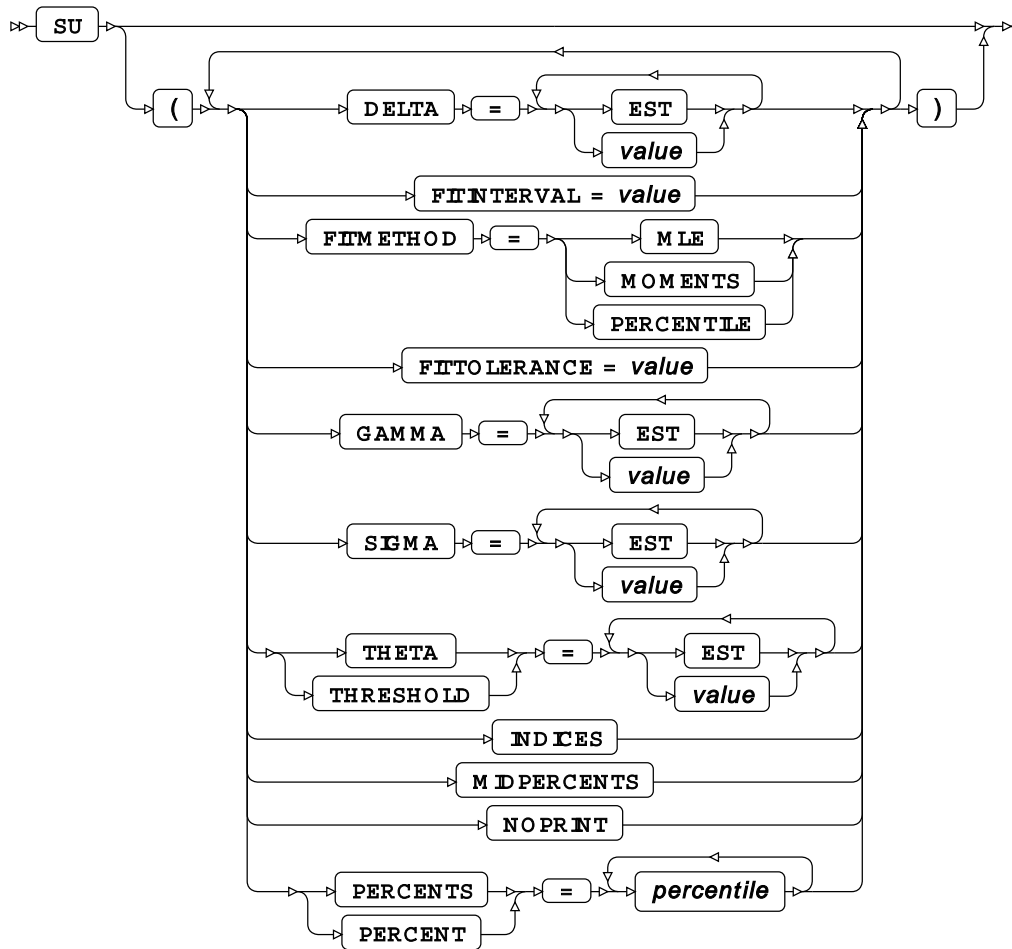
RAYLEIGH distribution



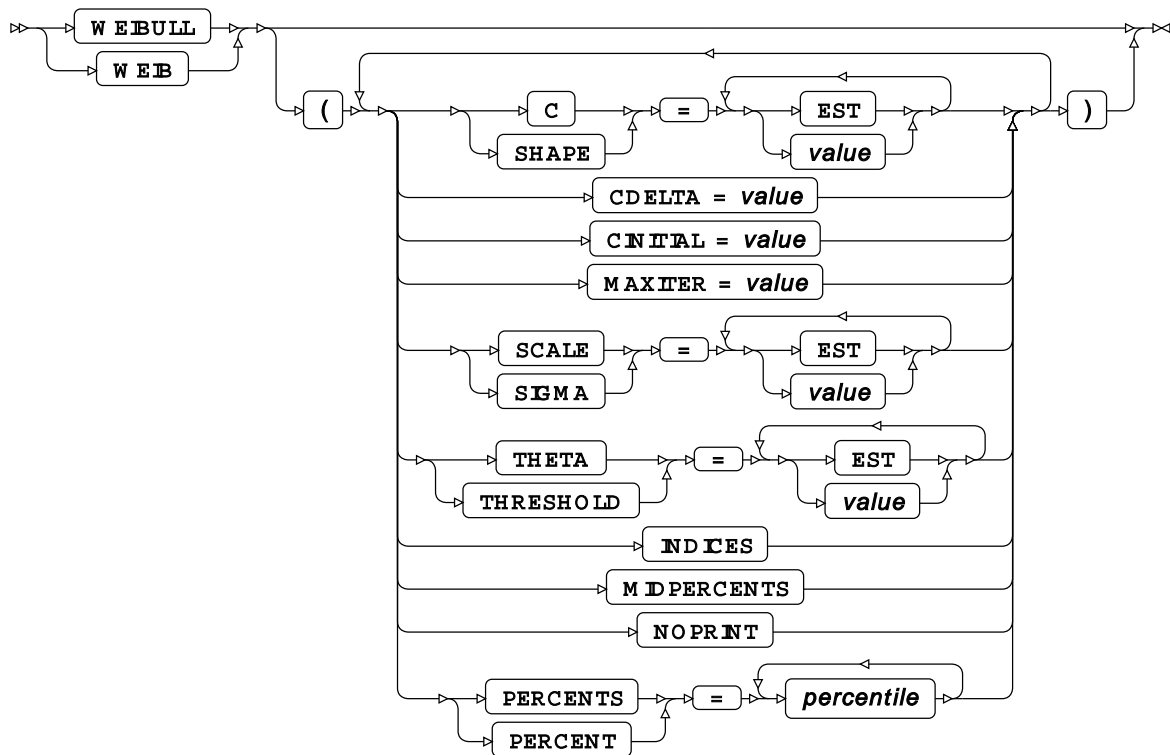
SB distribution



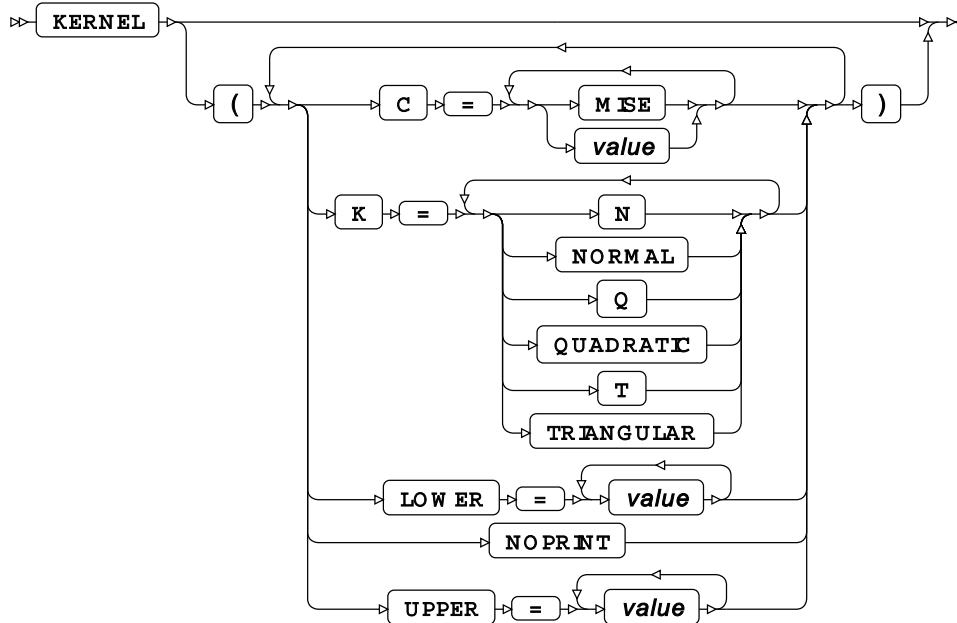
SU distribution



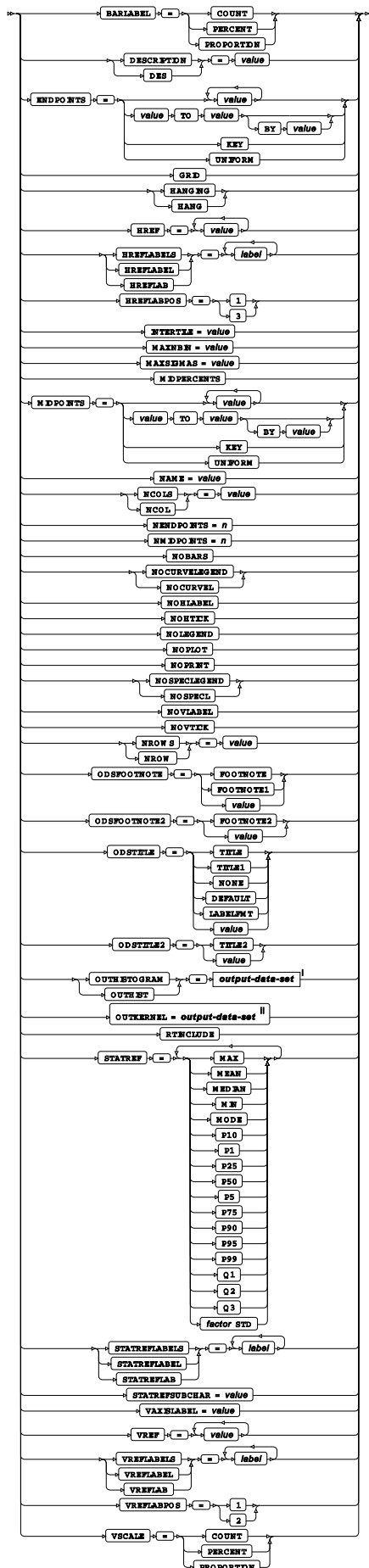
WEIBULL distribution



KERNEL option



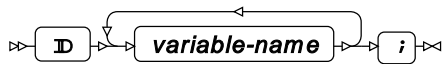
other-option



ⁱ See *Output dataset* [↗](#) (page 16).

ⁱⁱ See *Output dataset* [↗](#) (page 16).

ID

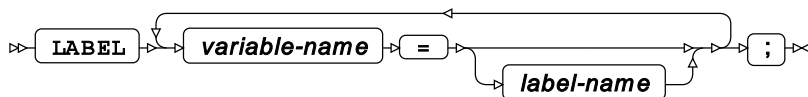


INFORMAT

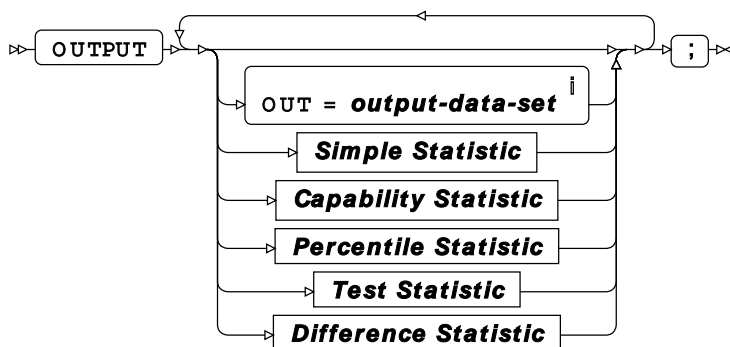


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL

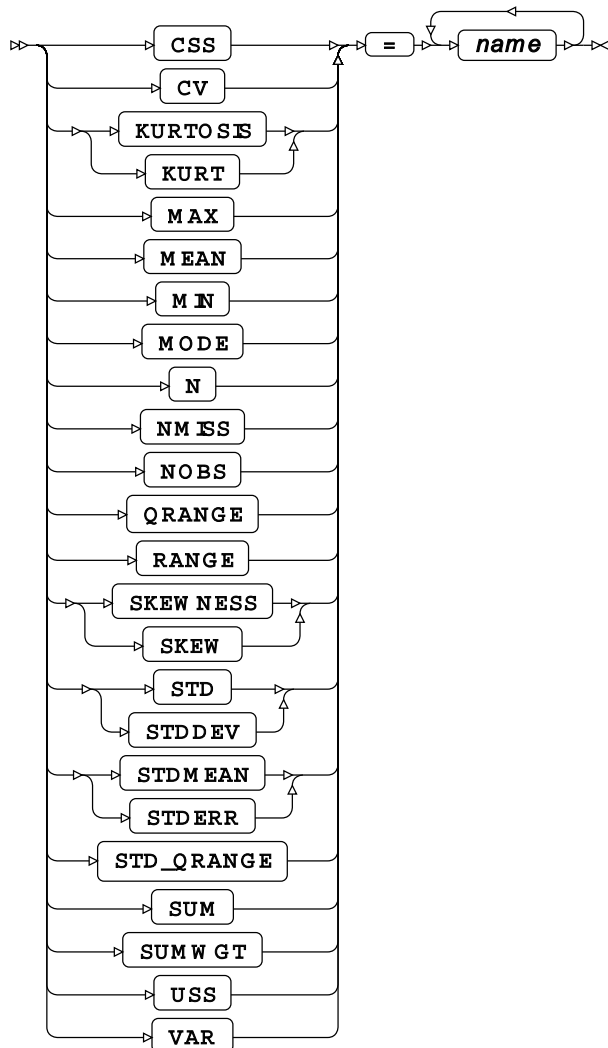


OUTPUT

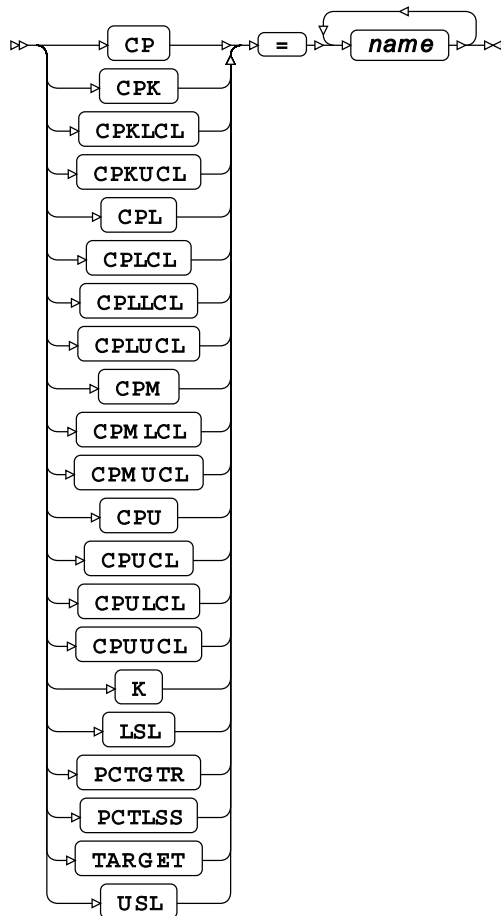


ⁱ See *Output dataset* [↗](#) (page 16).

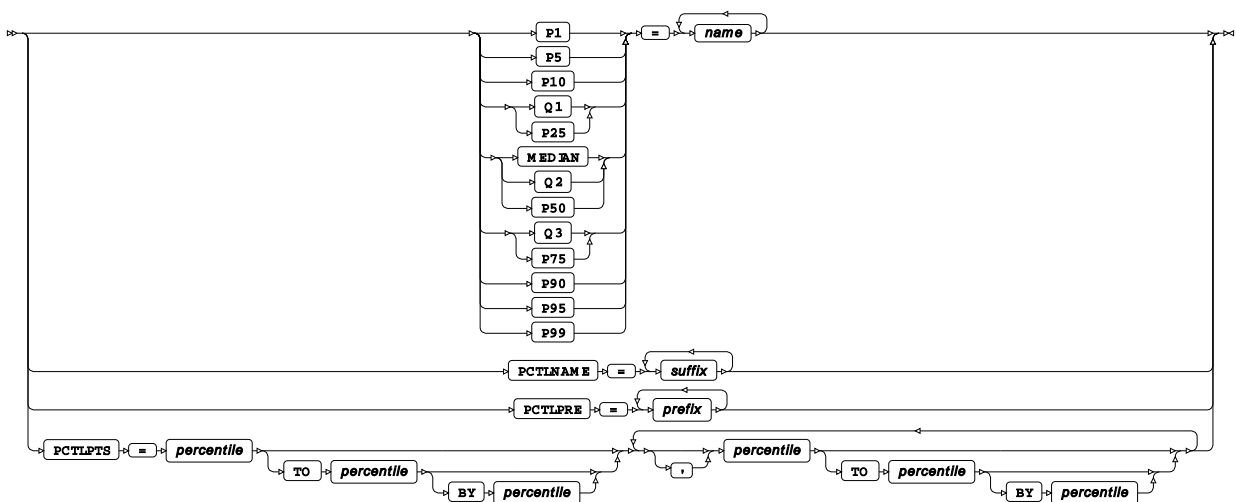
Simple Statistic



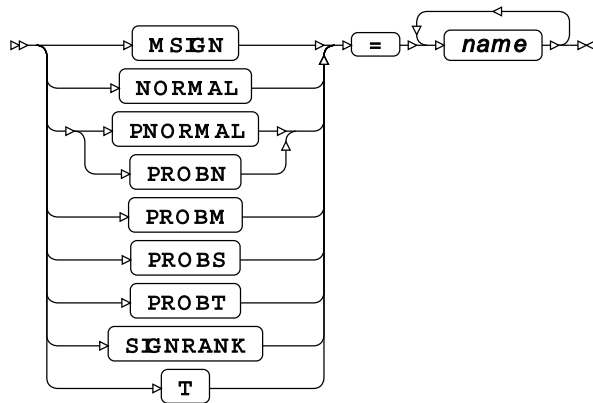
Capability Statistic



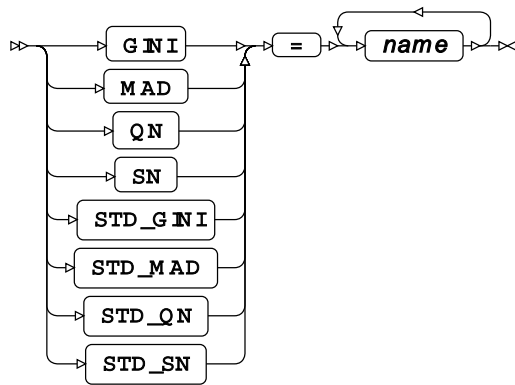
Percentile Statistic



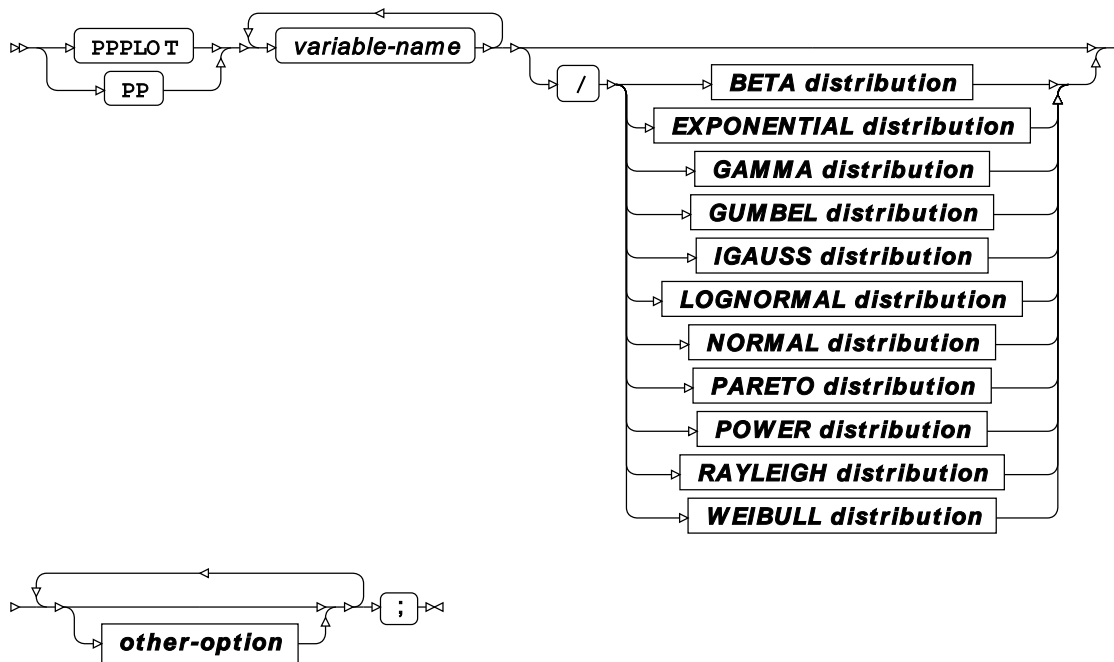
Test Statistic



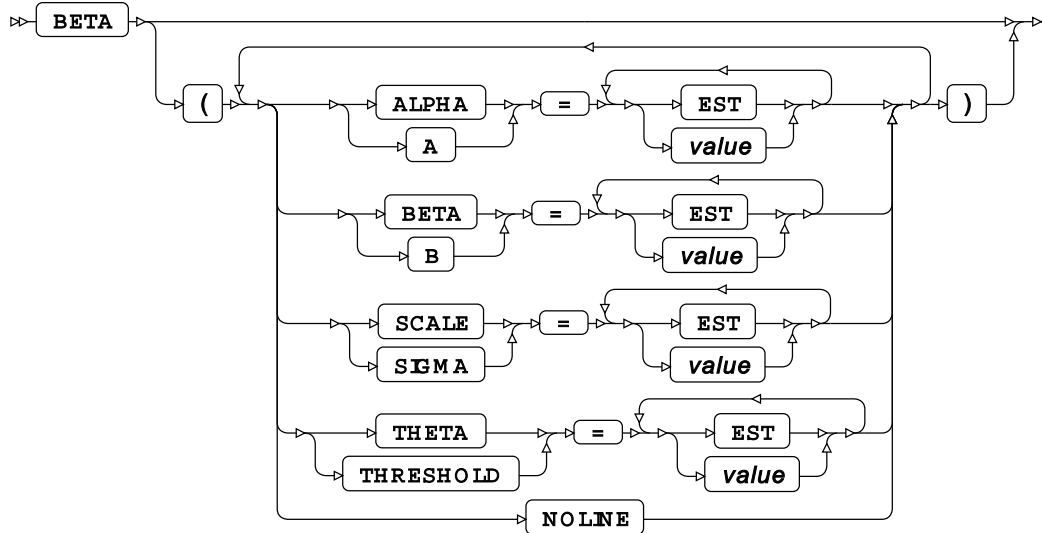
Difference Statistic



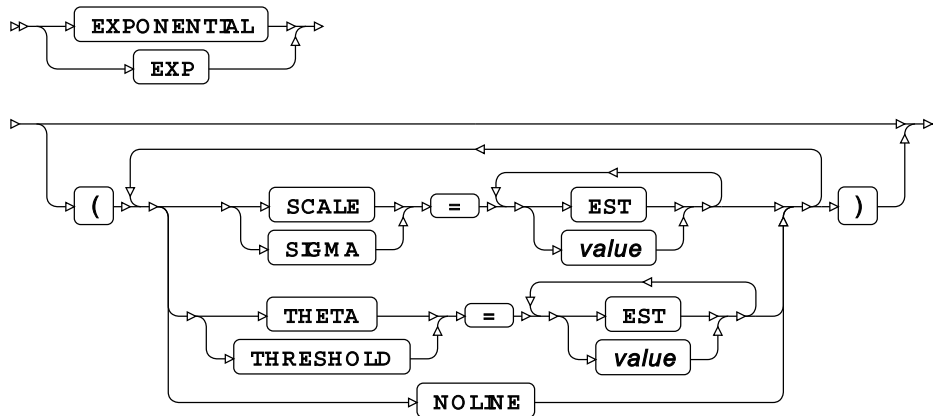
PPPLOT



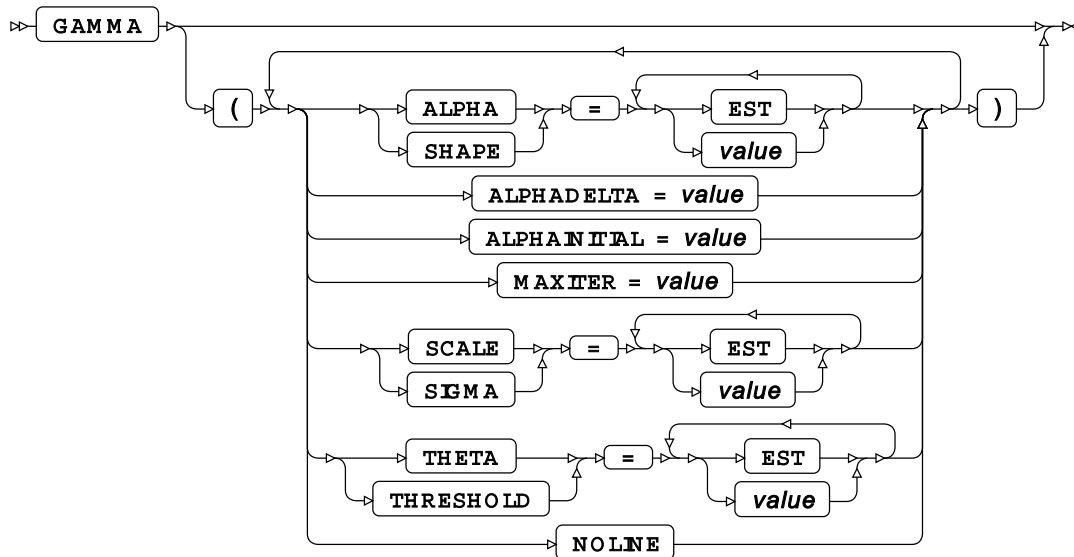
BETA distribution



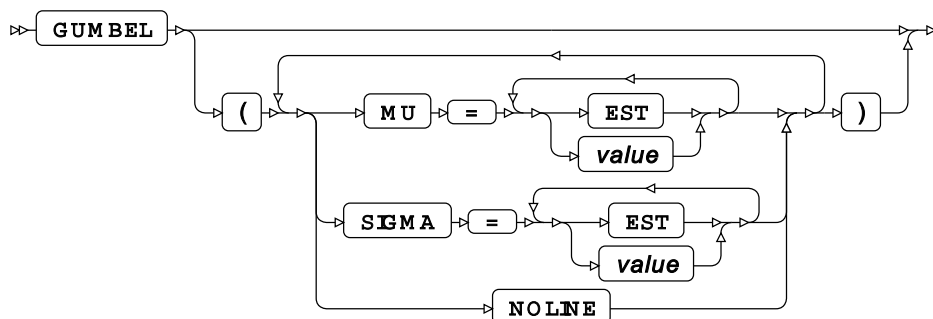
EXPONENTIAL distribution



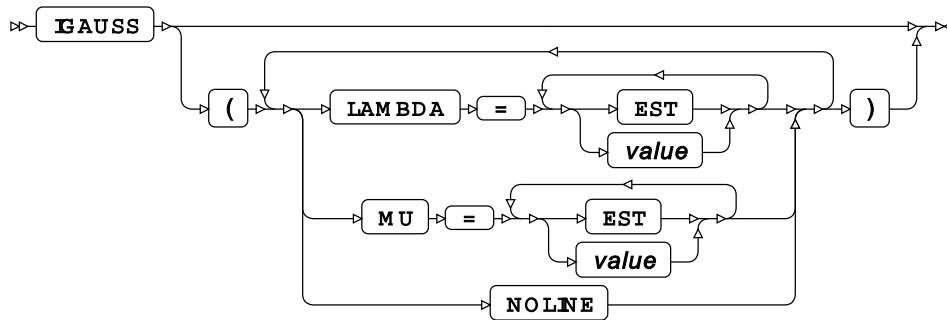
GAMMA distribution



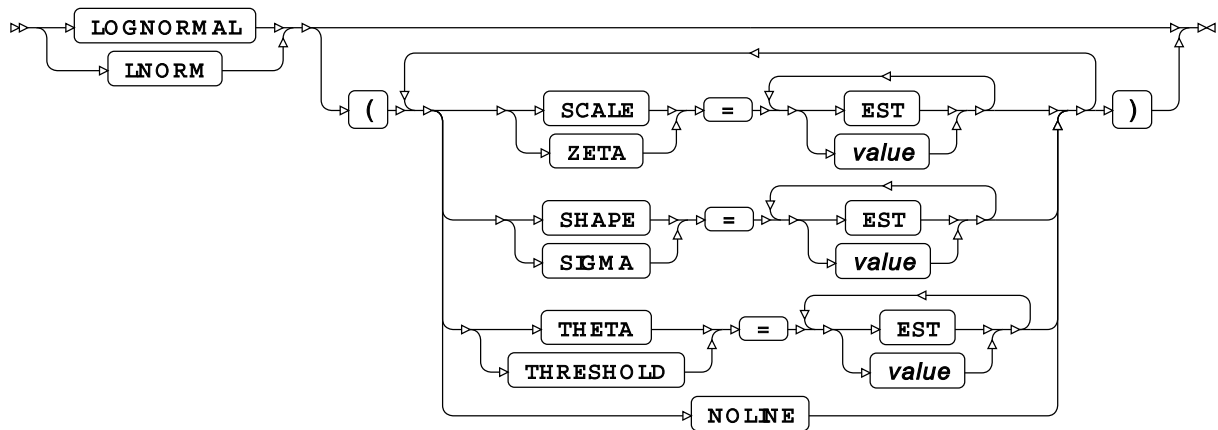
GUMBEL distribution



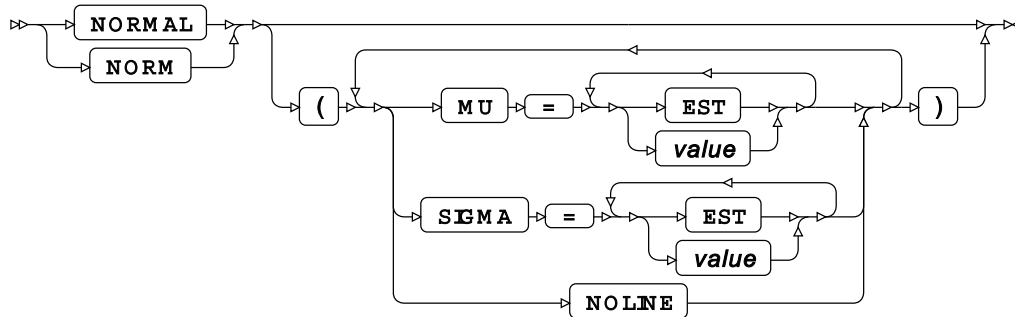
IGAUSS distribution



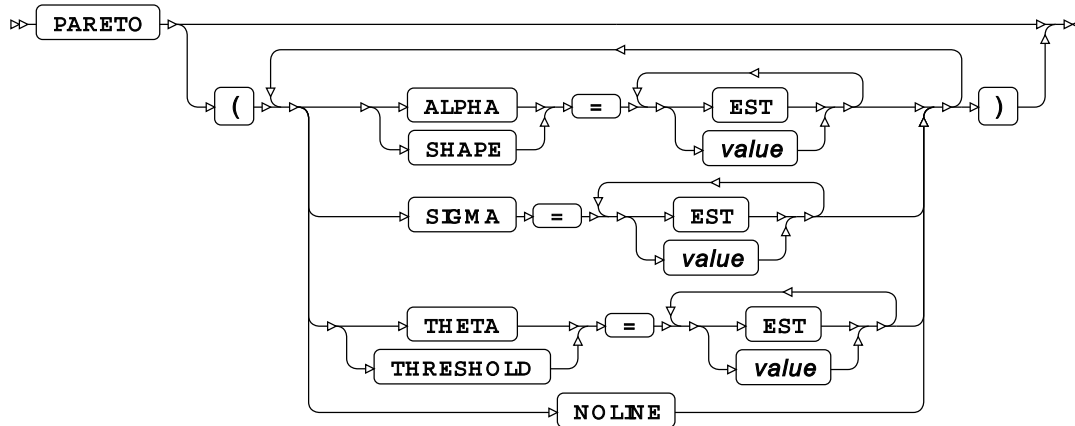
LOGNORMAL distribution



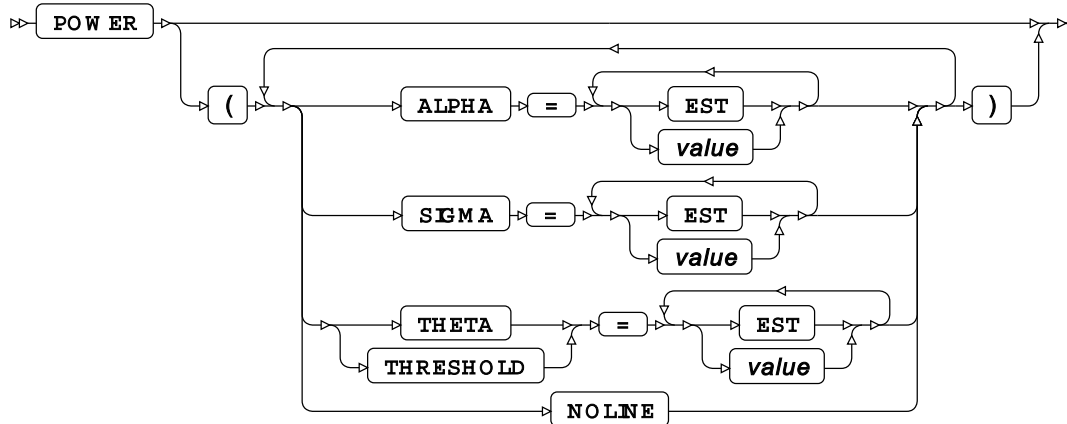
NORMAL distribution



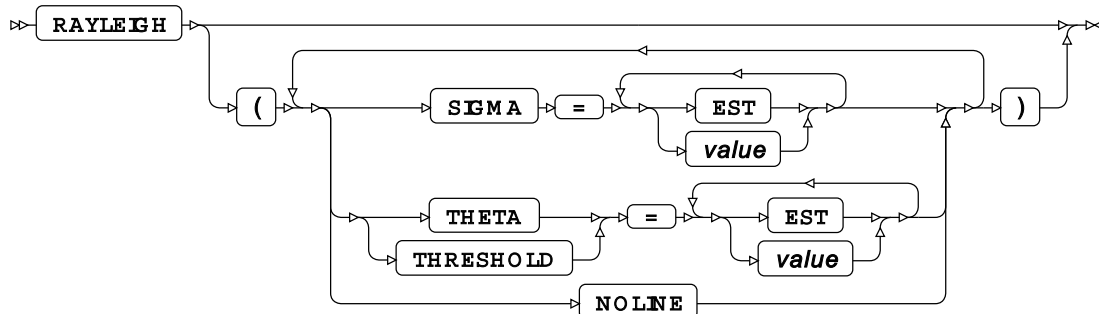
PARETO distribution



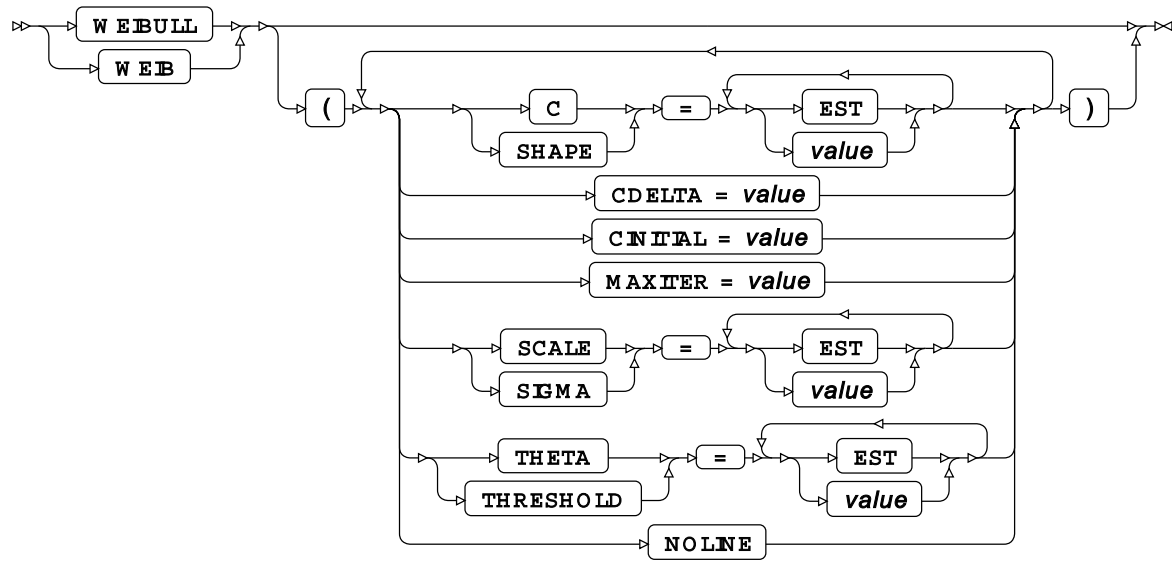
POWER distribution



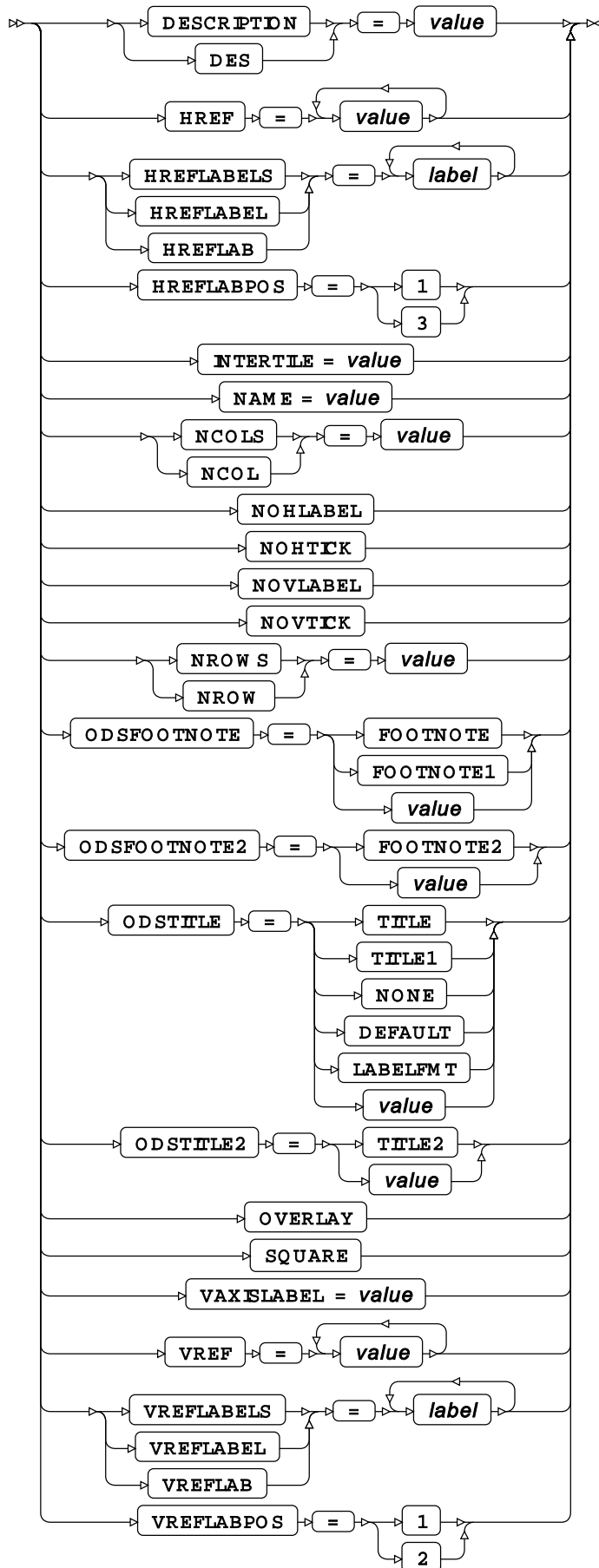
RAYLEIGH distribution



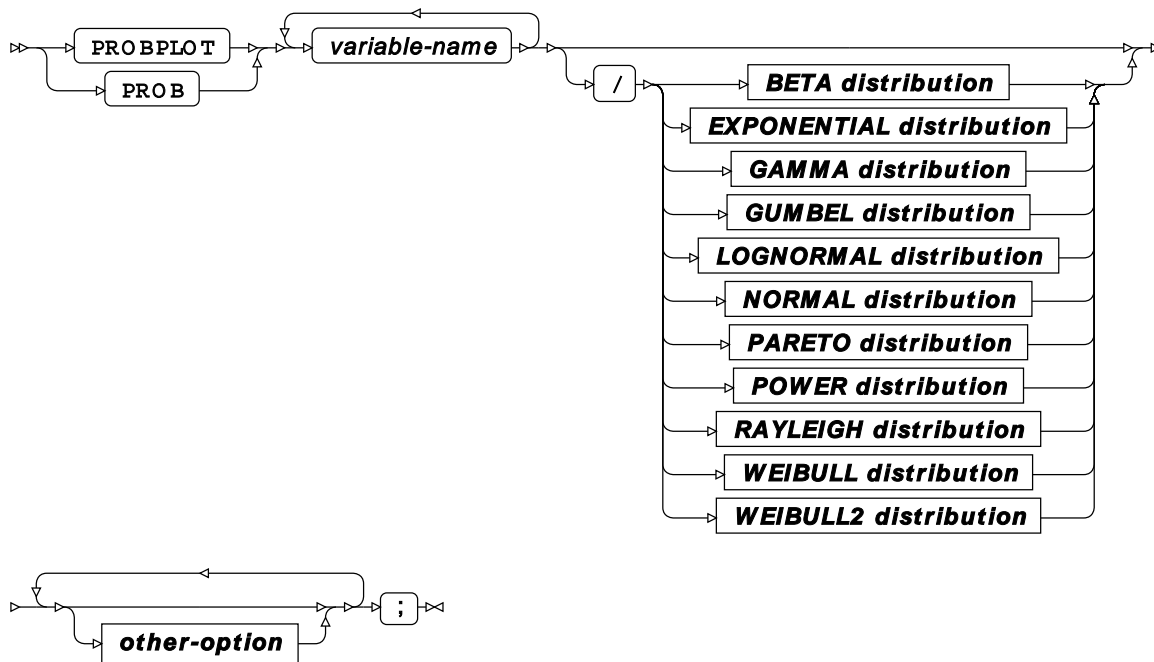
WEIBULL distribution



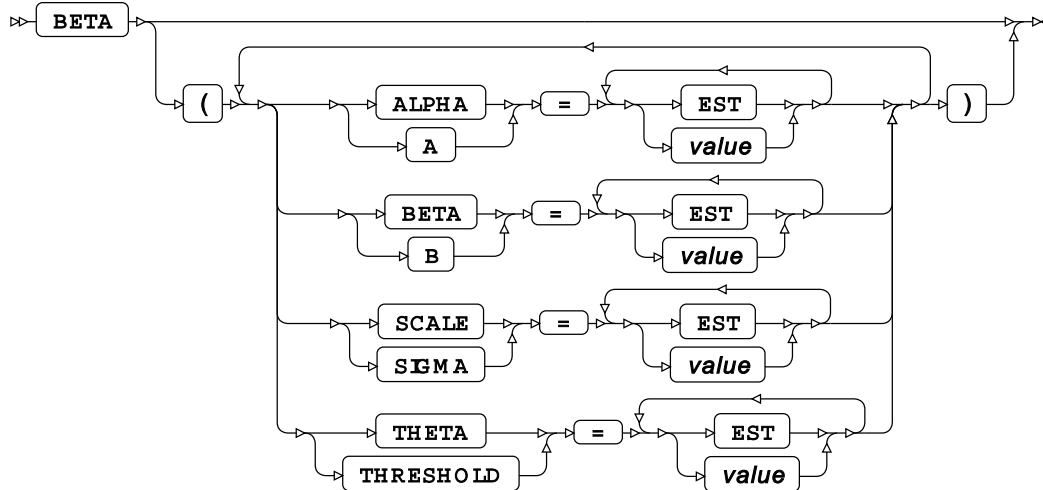
other-option



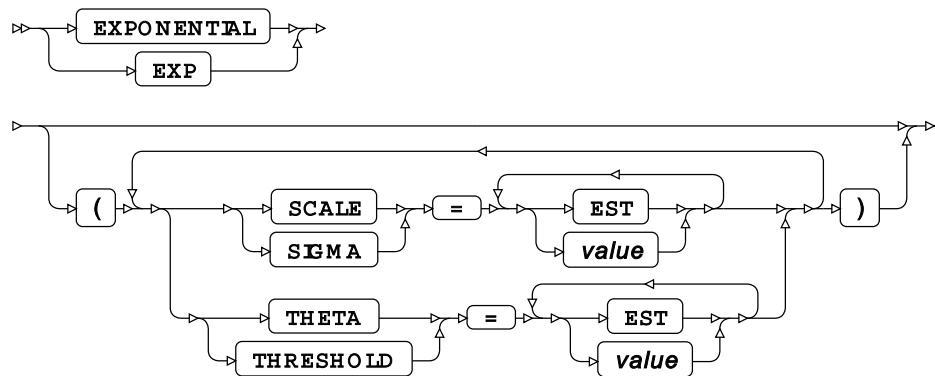
PROBPLOT



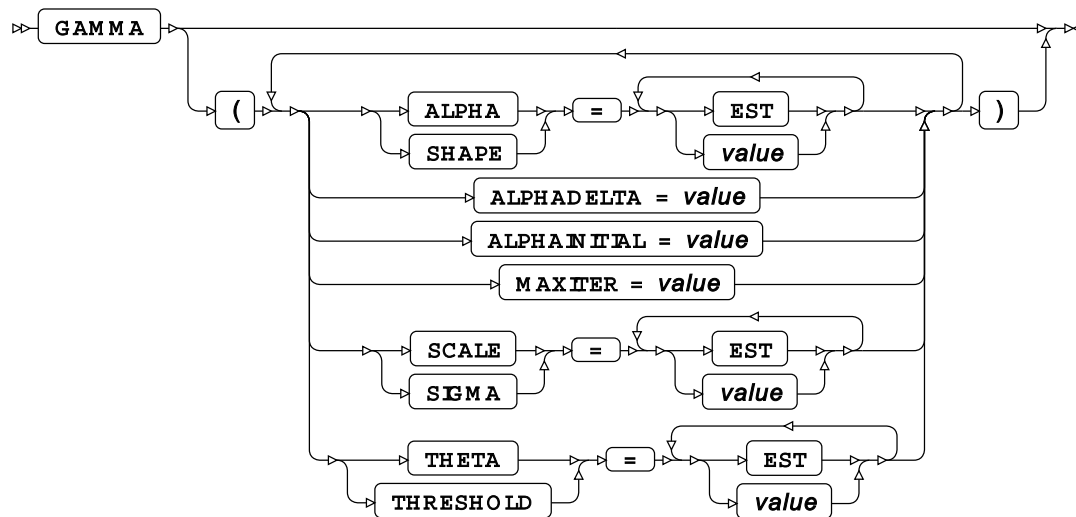
BETA distribution



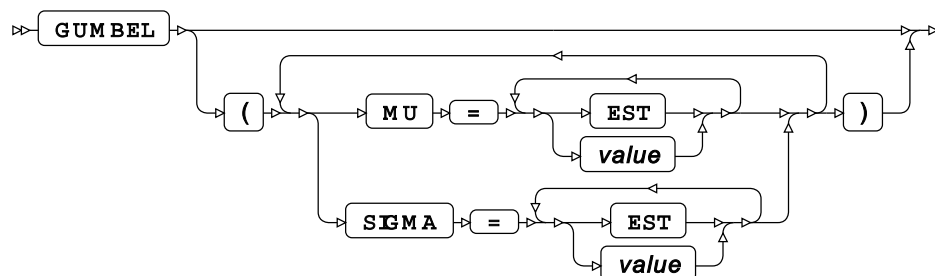
EXPONENTIAL distribution



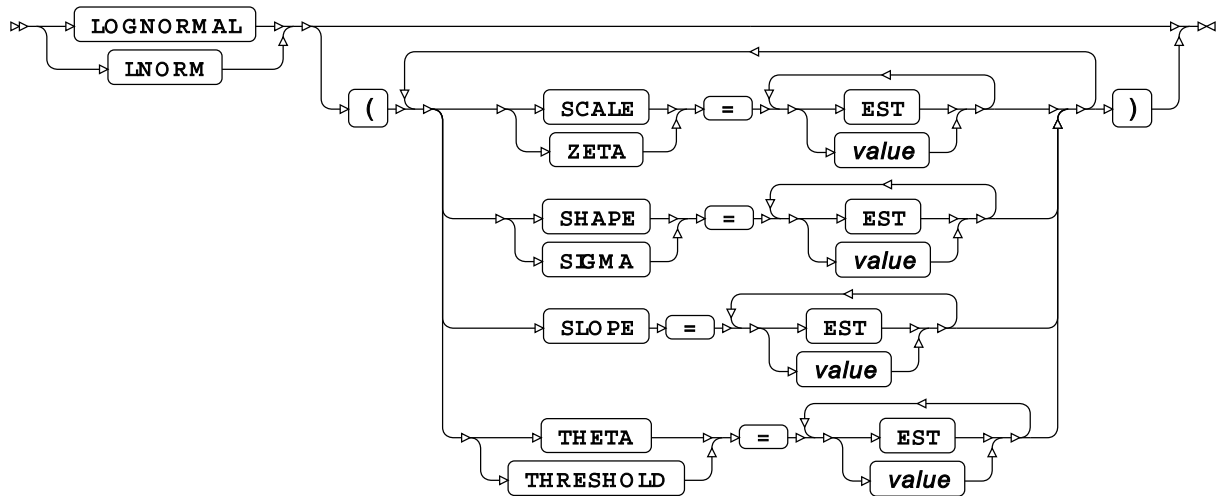
GAMMA distribution



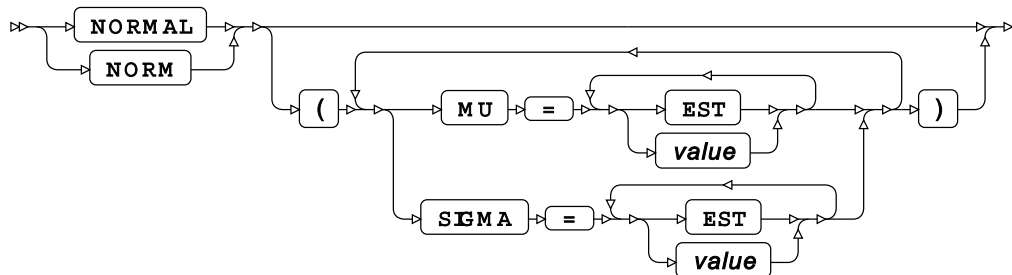
GUMBEL distribution



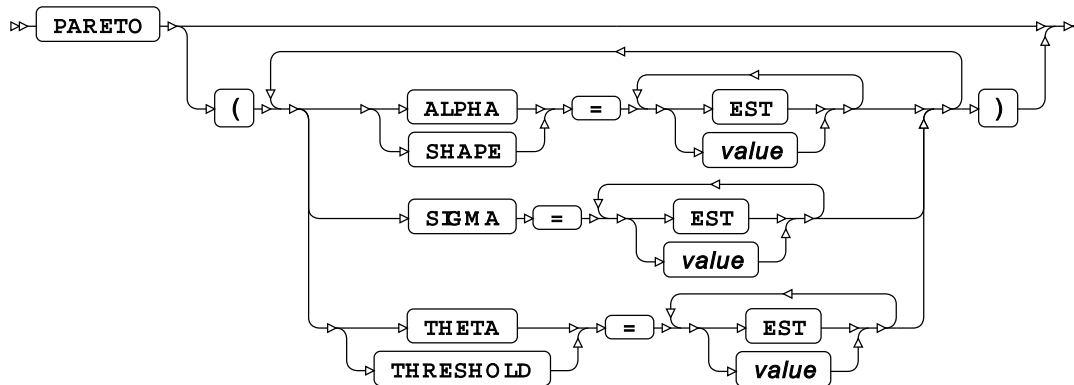
LOGNORMAL distribution



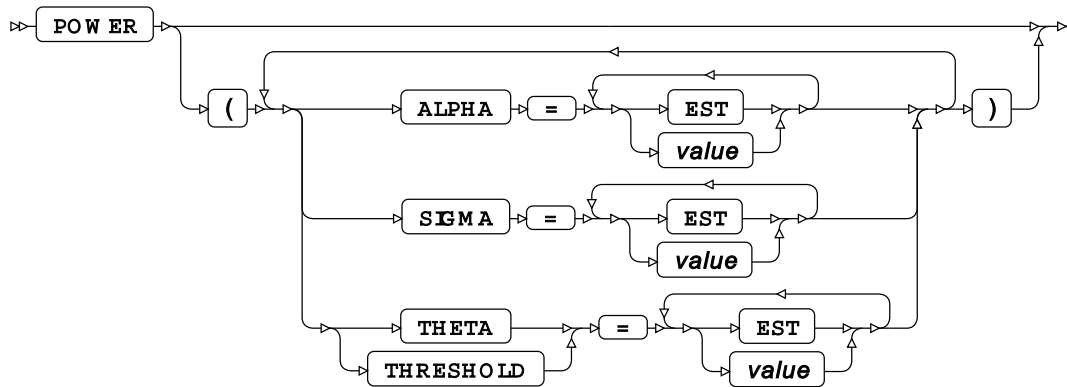
NORMAL distribution



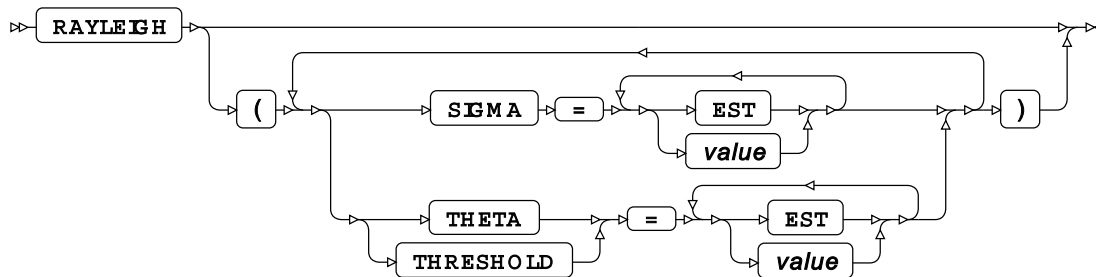
PARETO distribution



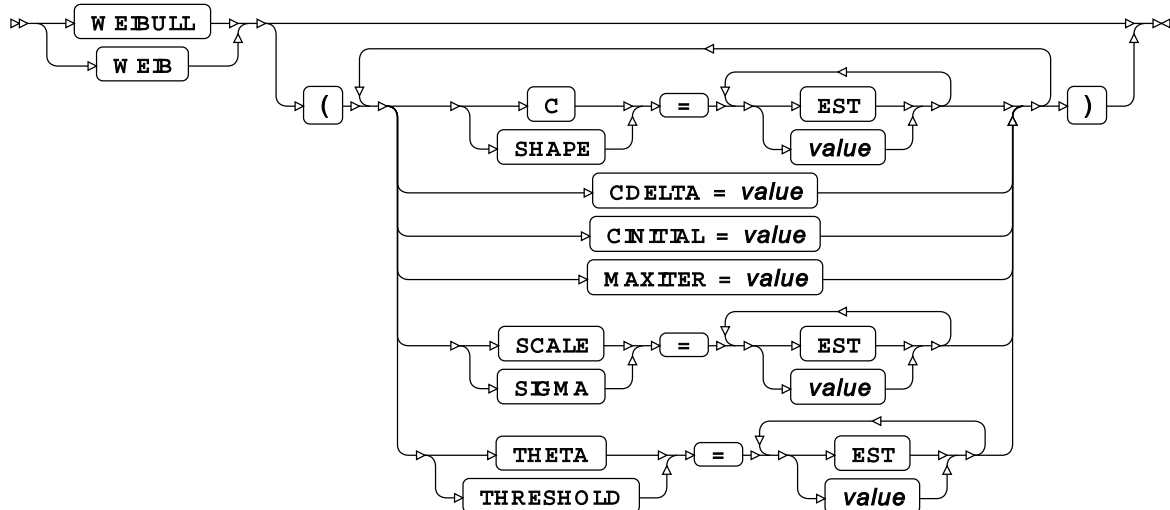
POWER distribution



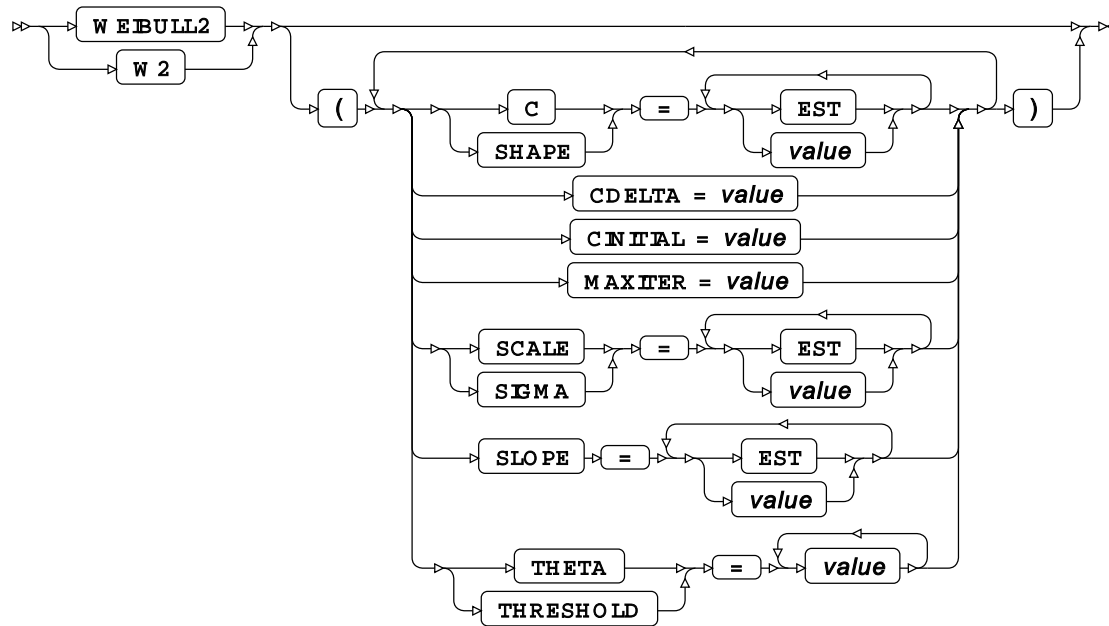
RAYLEIGH distribution



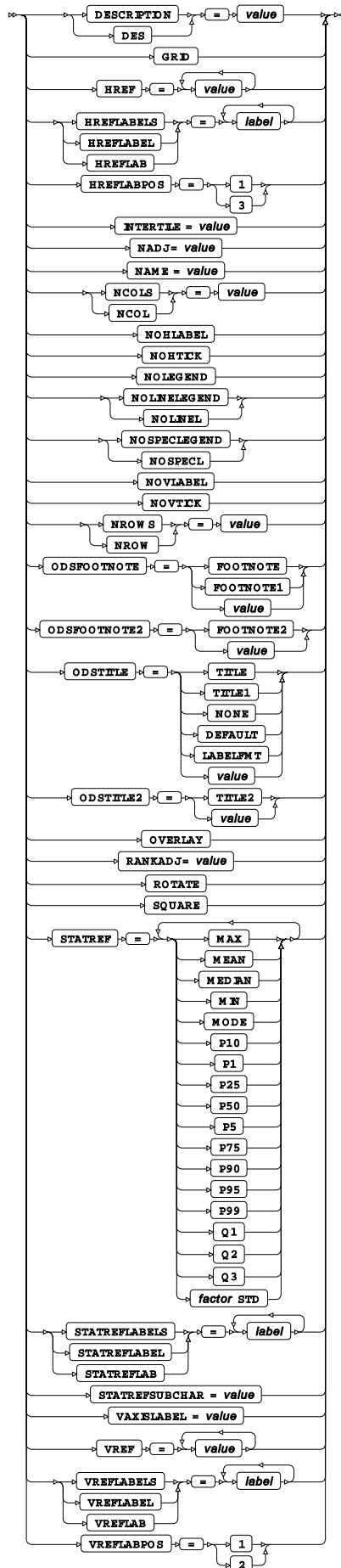
WEIBULL distribution



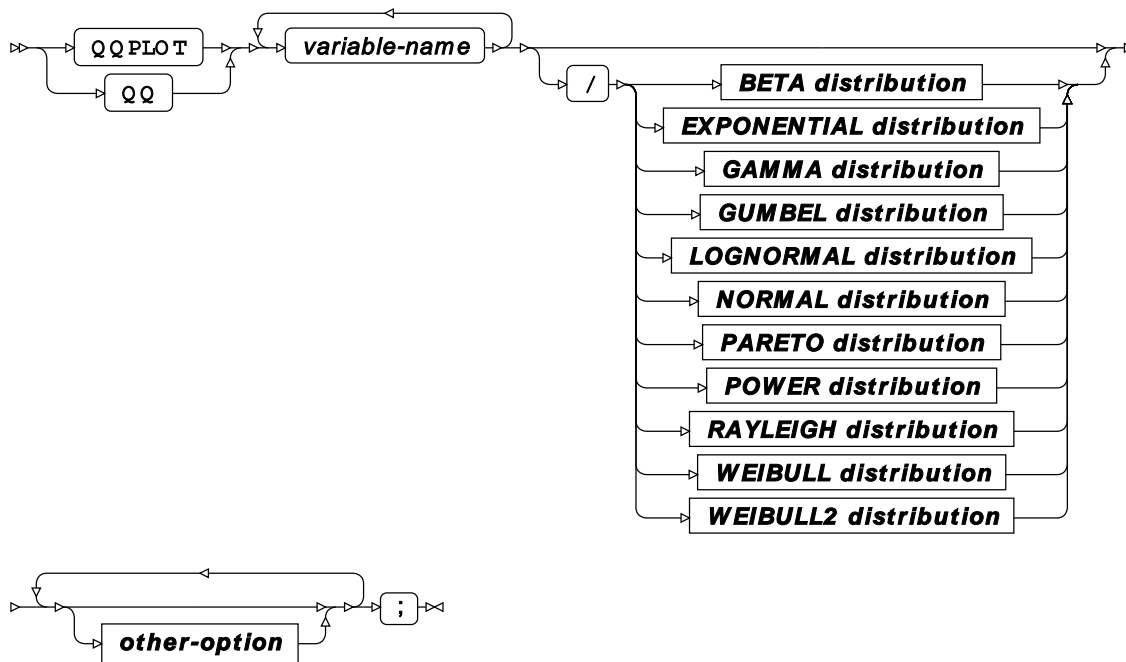
WEIBULL2 distribution



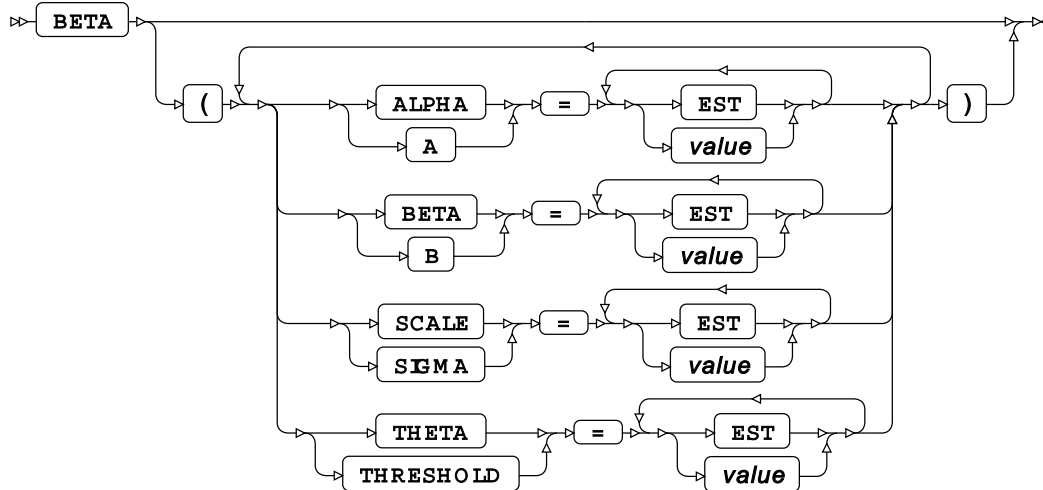
other-option



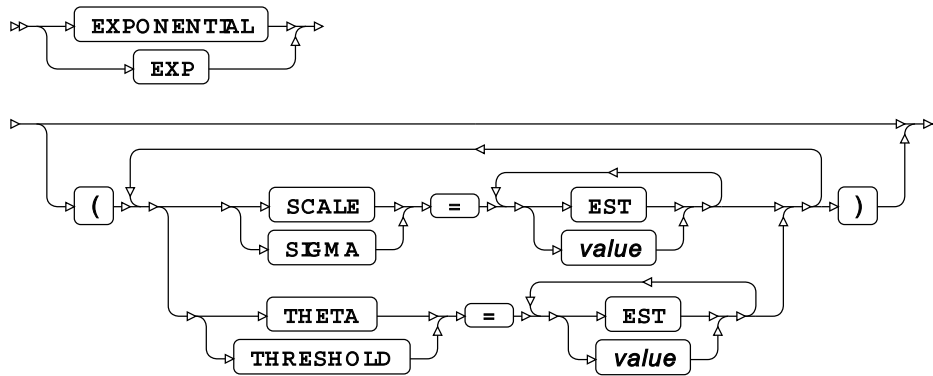
QQPLOT



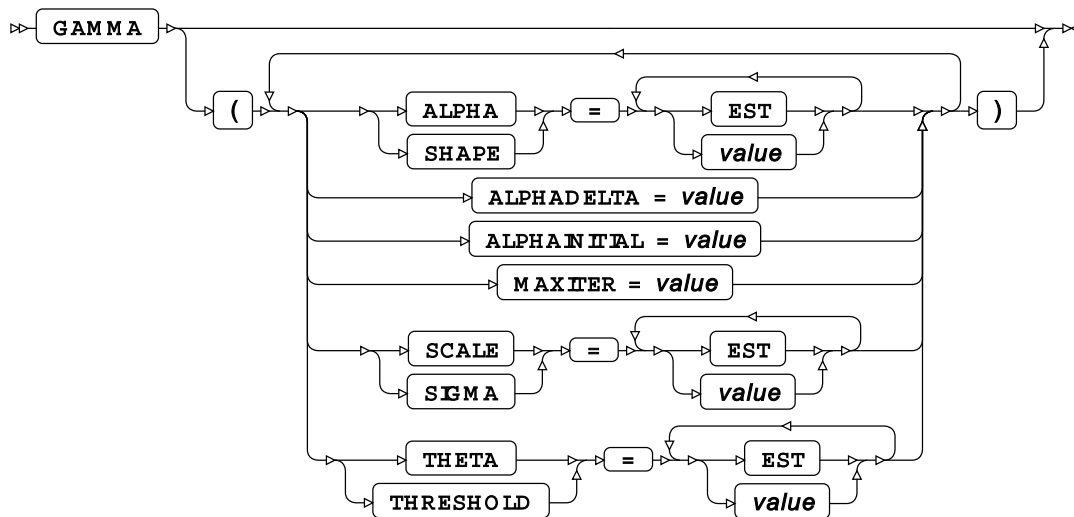
BETA distribution



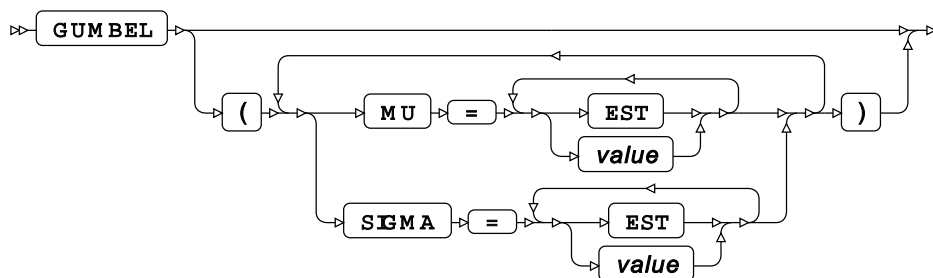
EXPONENTIAL distribution



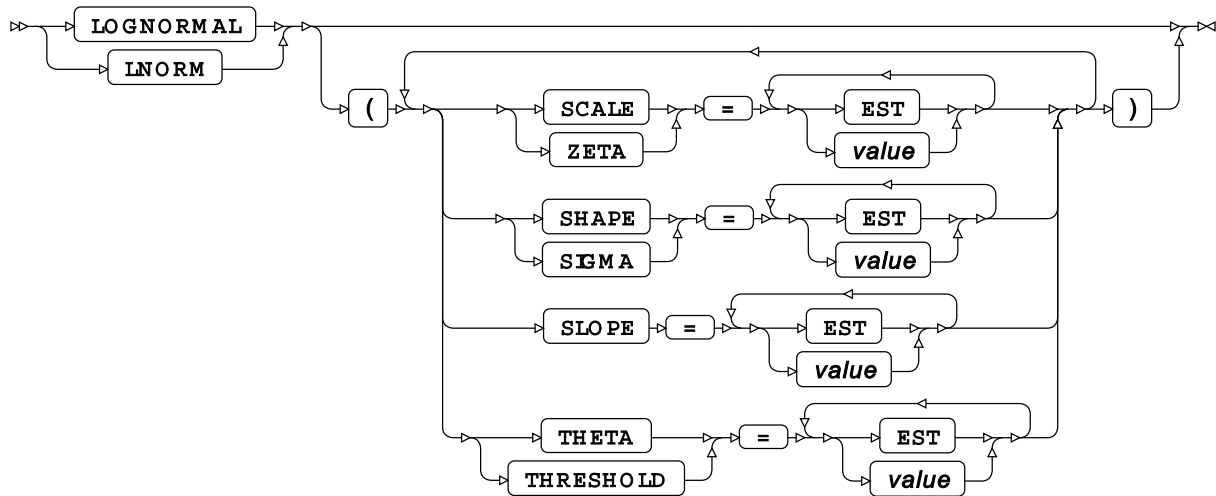
GAMMA distribution



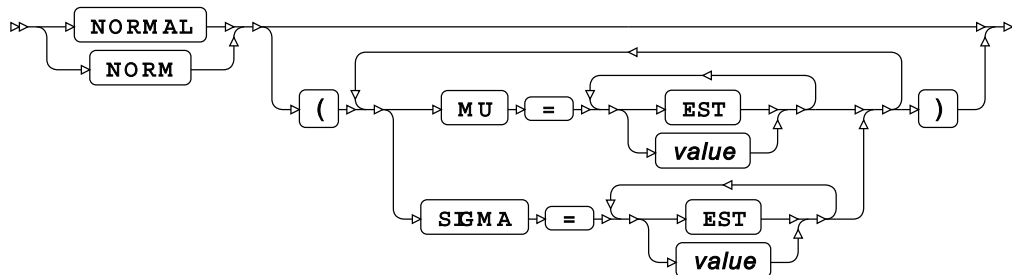
GUMBEL distribution



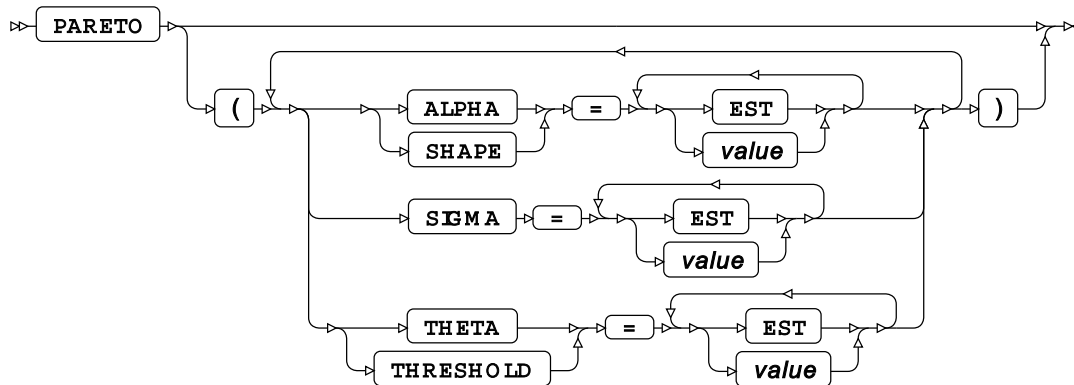
LOGNORMAL distribution



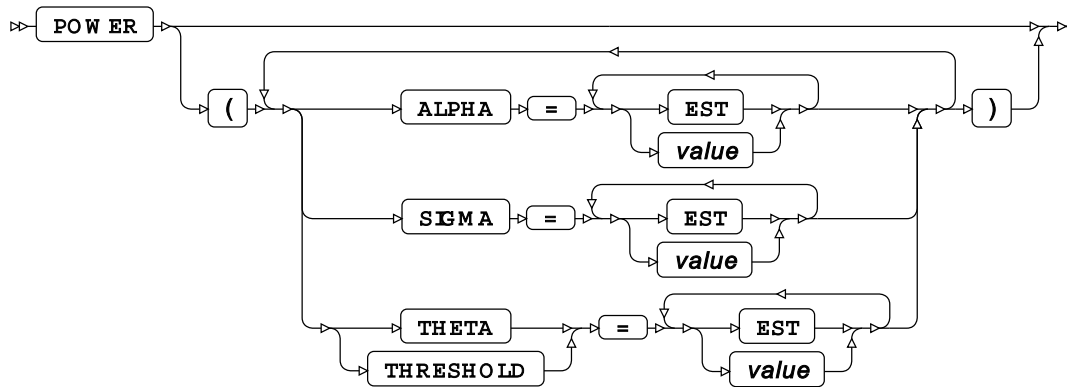
NORMAL distribution



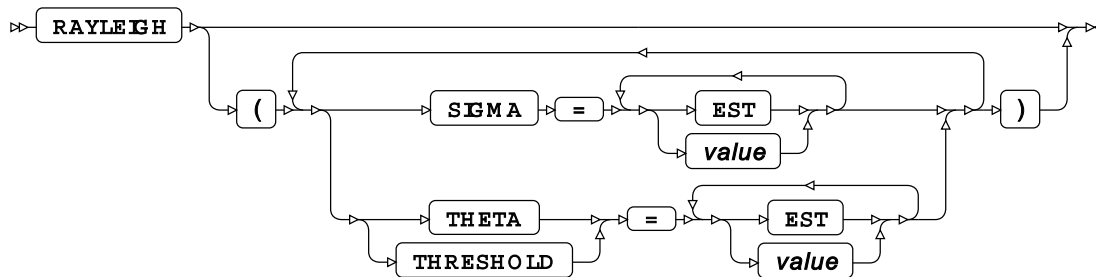
PARETO distribution



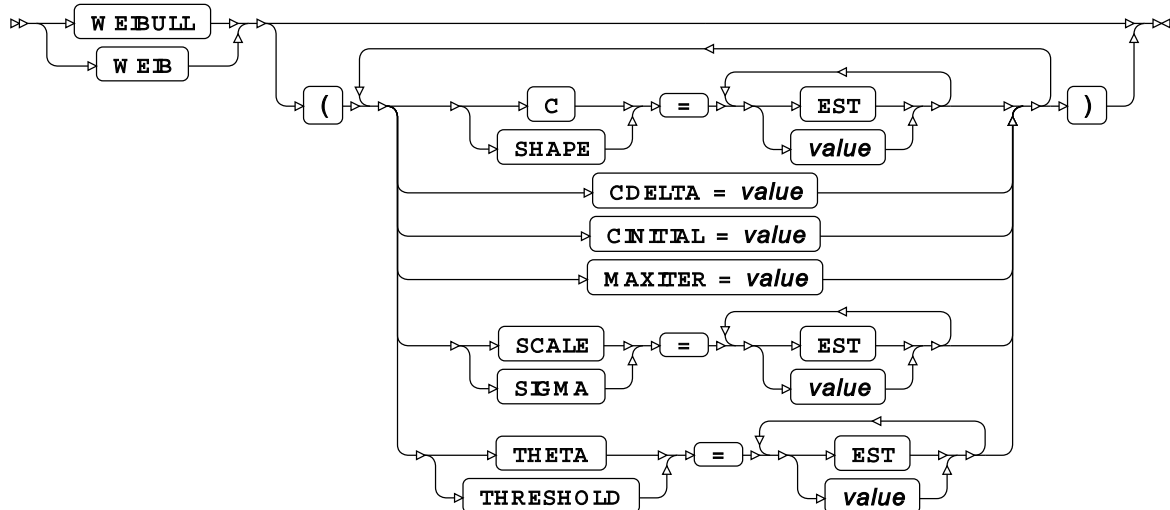
POWER distribution



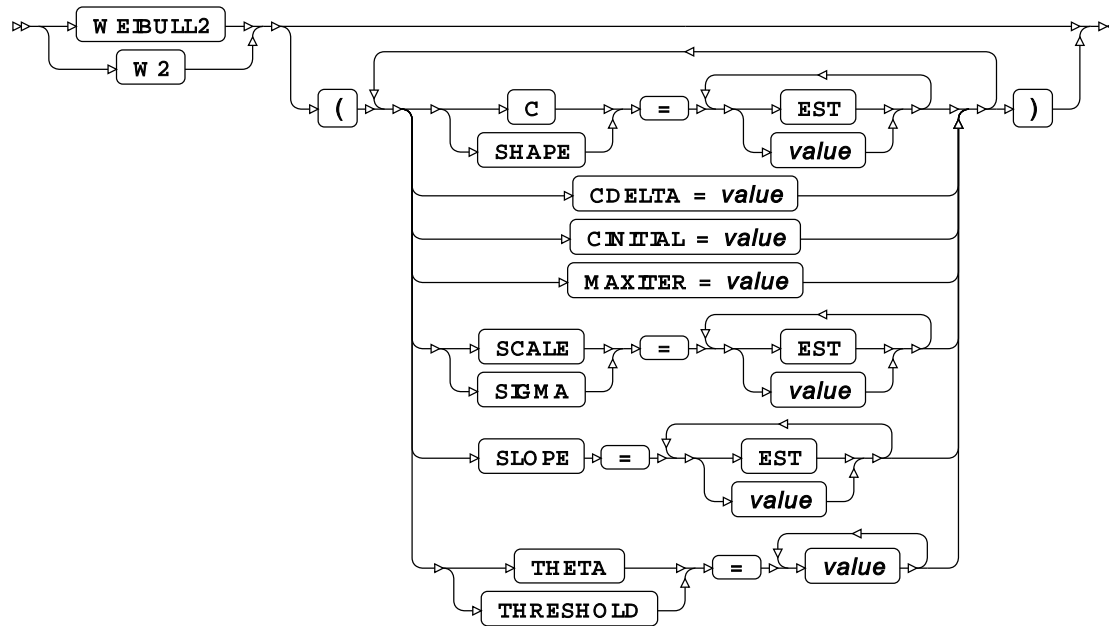
RAYLEIGH distribution



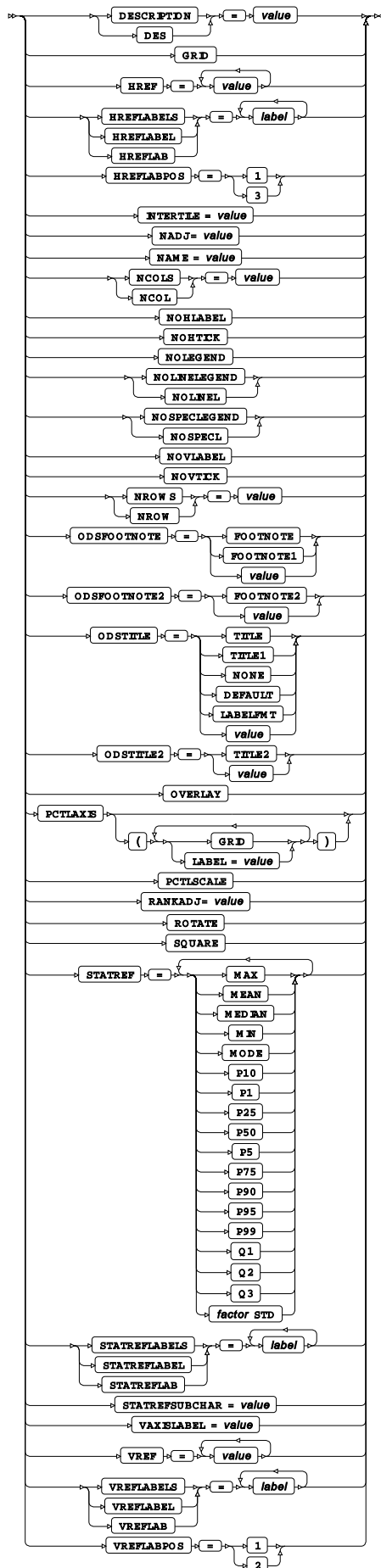
WEIBULL distribution



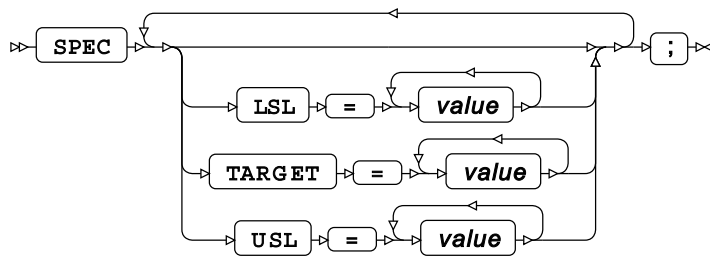
WEIBULL2 distribution



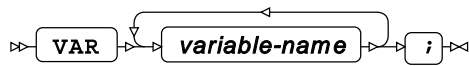
other-option



SPEC



VAR



WEIGHT



WHERE



WPS Timeseries

Timeseries procedures

ARIMA procedure

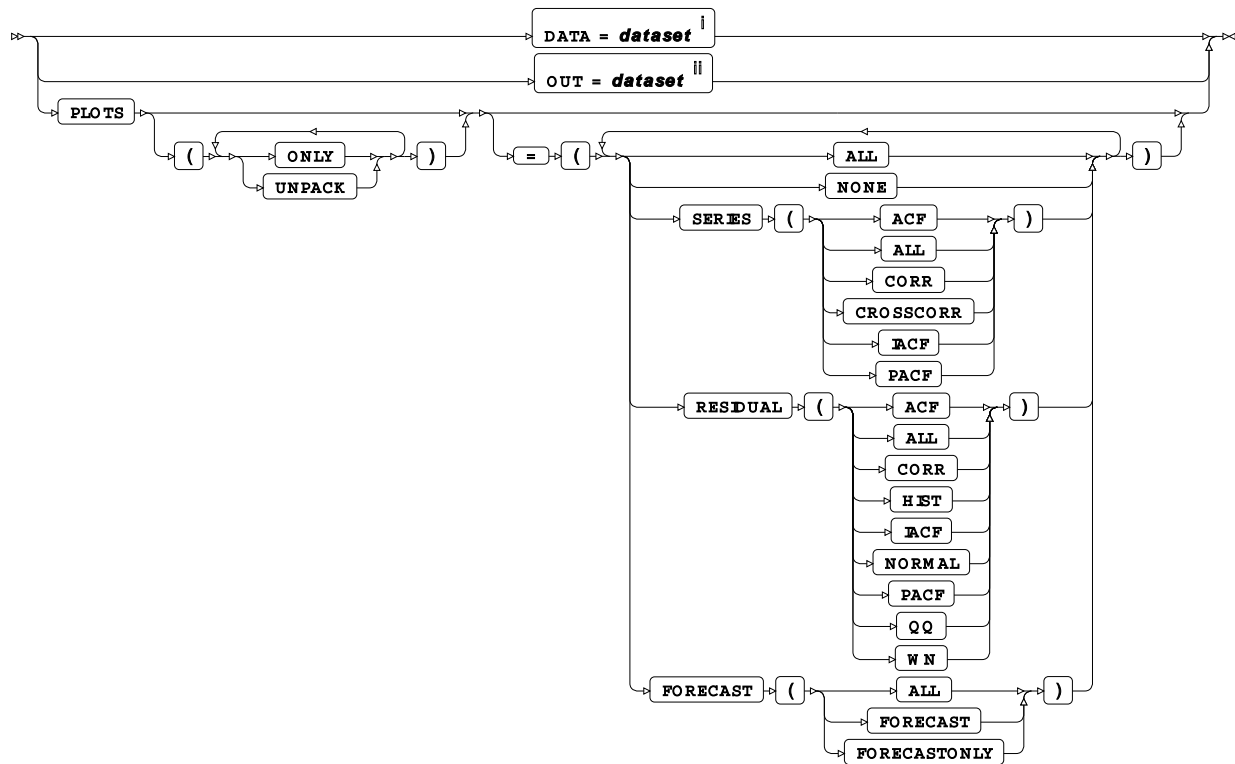
Supported statements

- *PROC ARIMA* [↗](#) (page 3441)
- *ATTRIB* [↗](#) (page 3442)
- *BY* [↗](#) (page 3443)
- *FORMAT* [↗](#) (page 3443)
- *IDENTIFY* [↗](#) (page 3443)
- *INFORMAT* [↗](#) (page 3444)
- *ESTIMATE* [↗](#) (page 3445)
- *FORECAST* [↗](#) (page 3447)
- *LABEL* [↗](#) (page 3448)
- *WHERE* [↗](#) (page 3448)

PROC ARIMA



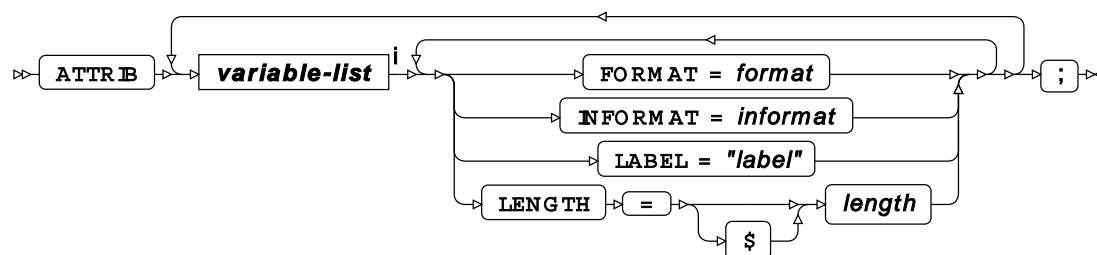
ARIMA-options



ⁱ See *Input dataset* [↗](#) (page 16).

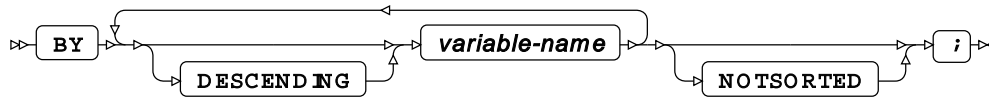
ⁱⁱ See *Output dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY

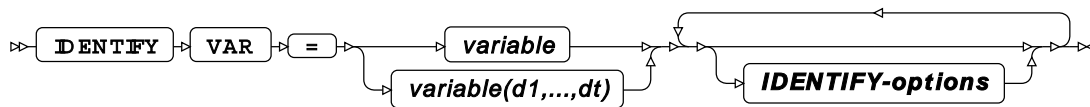


FORMAT

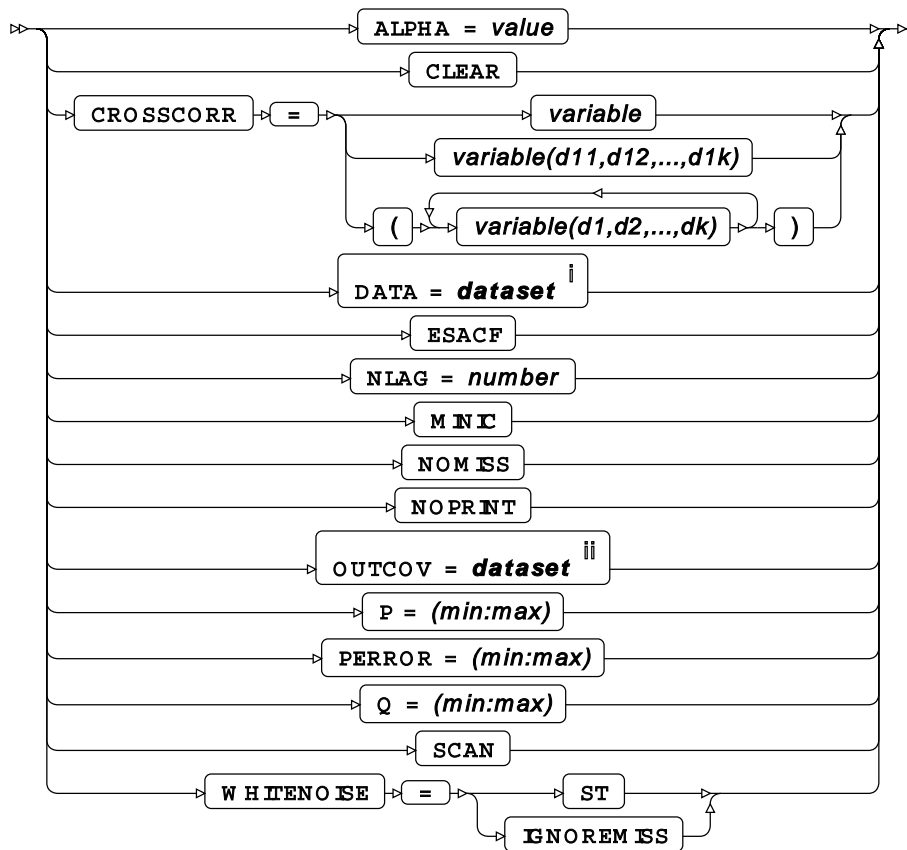


ⁱ See [Variable Lists](#) (page 32).

IDENTIFY



IDENTIFY-options



ⁱ See *Input dataset* [↗](#) (page 16).

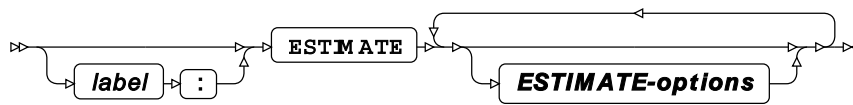
ⁱⁱ See *Output dataset* [↗](#) (page 16).

INFORMAT

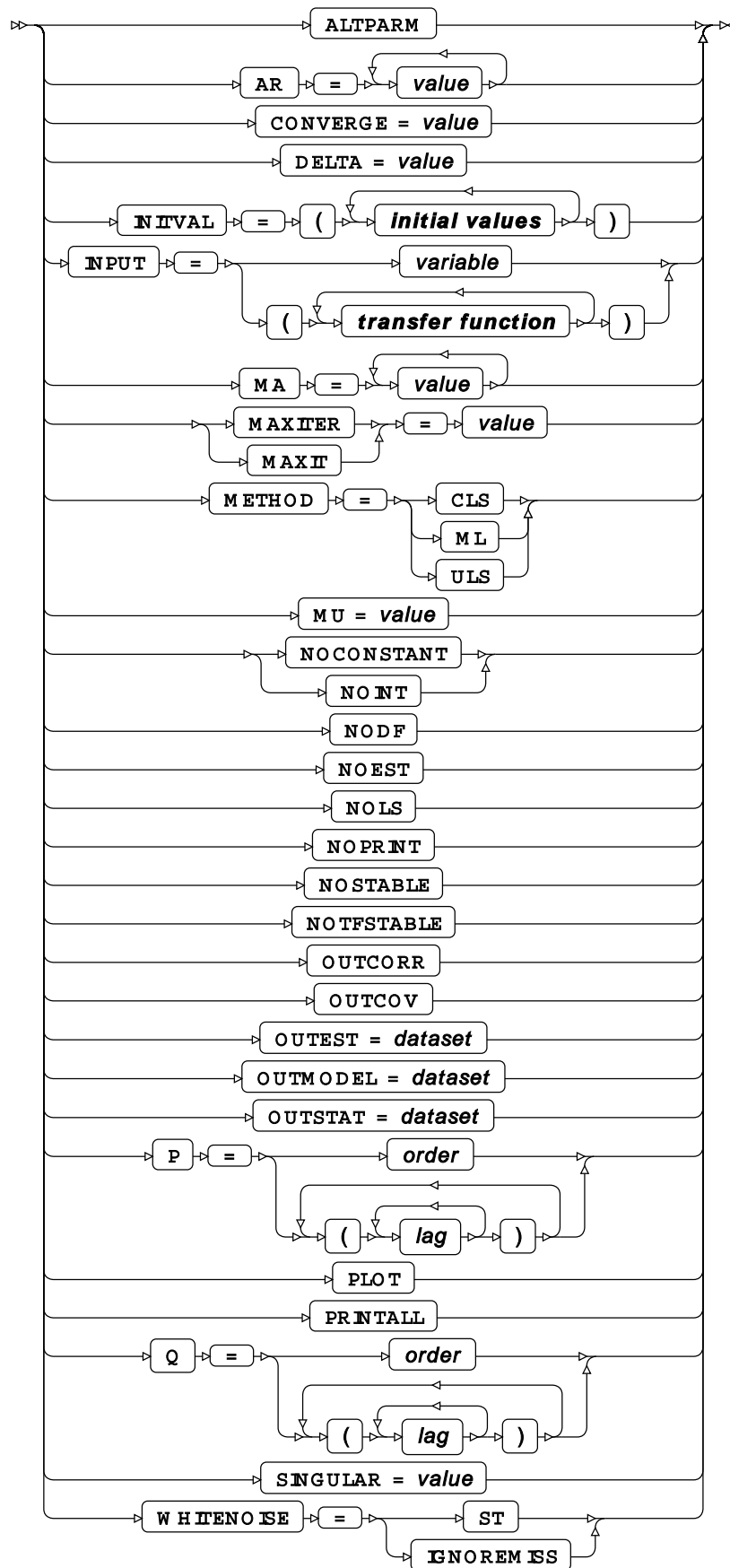


ⁱ See *Variable Lists* [↗](#) (page 32).

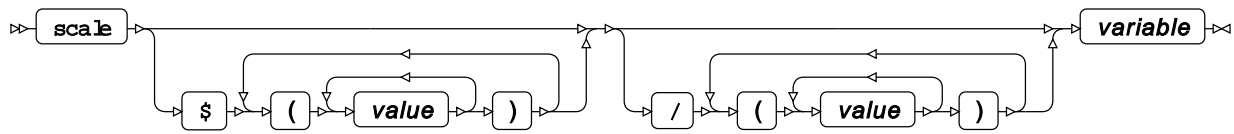
ESTIMATE



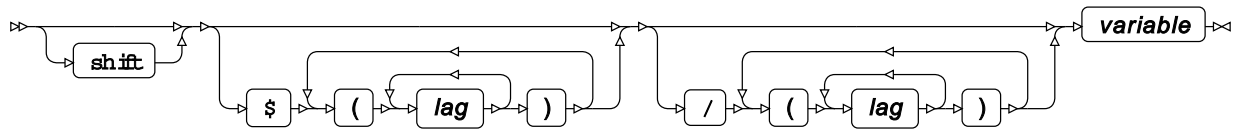
ESTIMATE-options



initial values



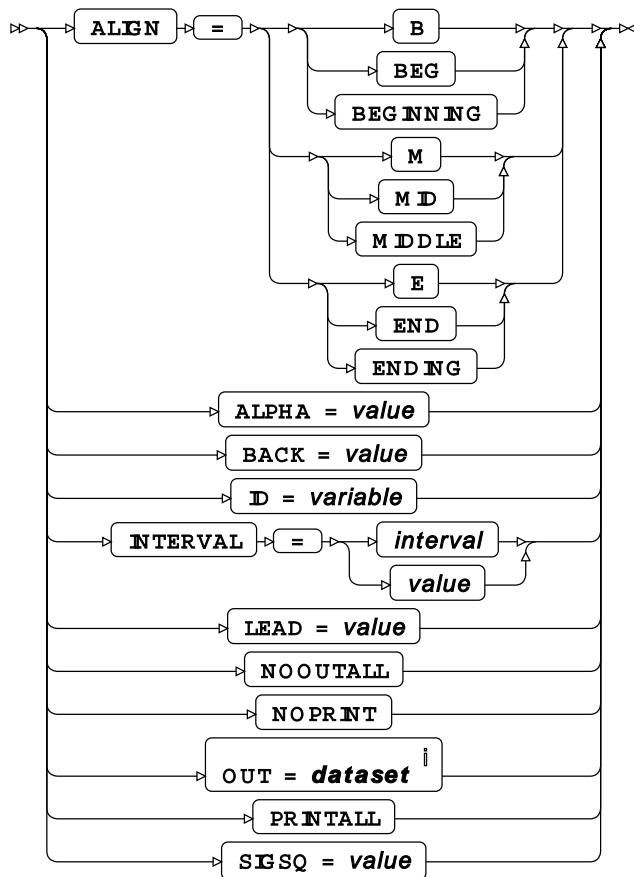
transfer function



FORECAST

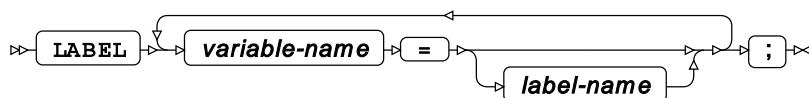


FORECAST-options

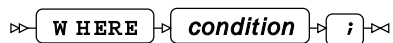


ⁱ See *Output dataset* [↗](#) (page 16).

LABEL



WHERE

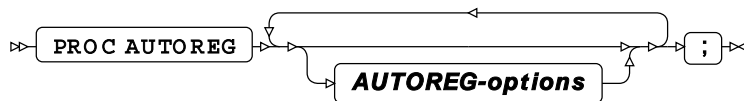


AUTOREG procedure

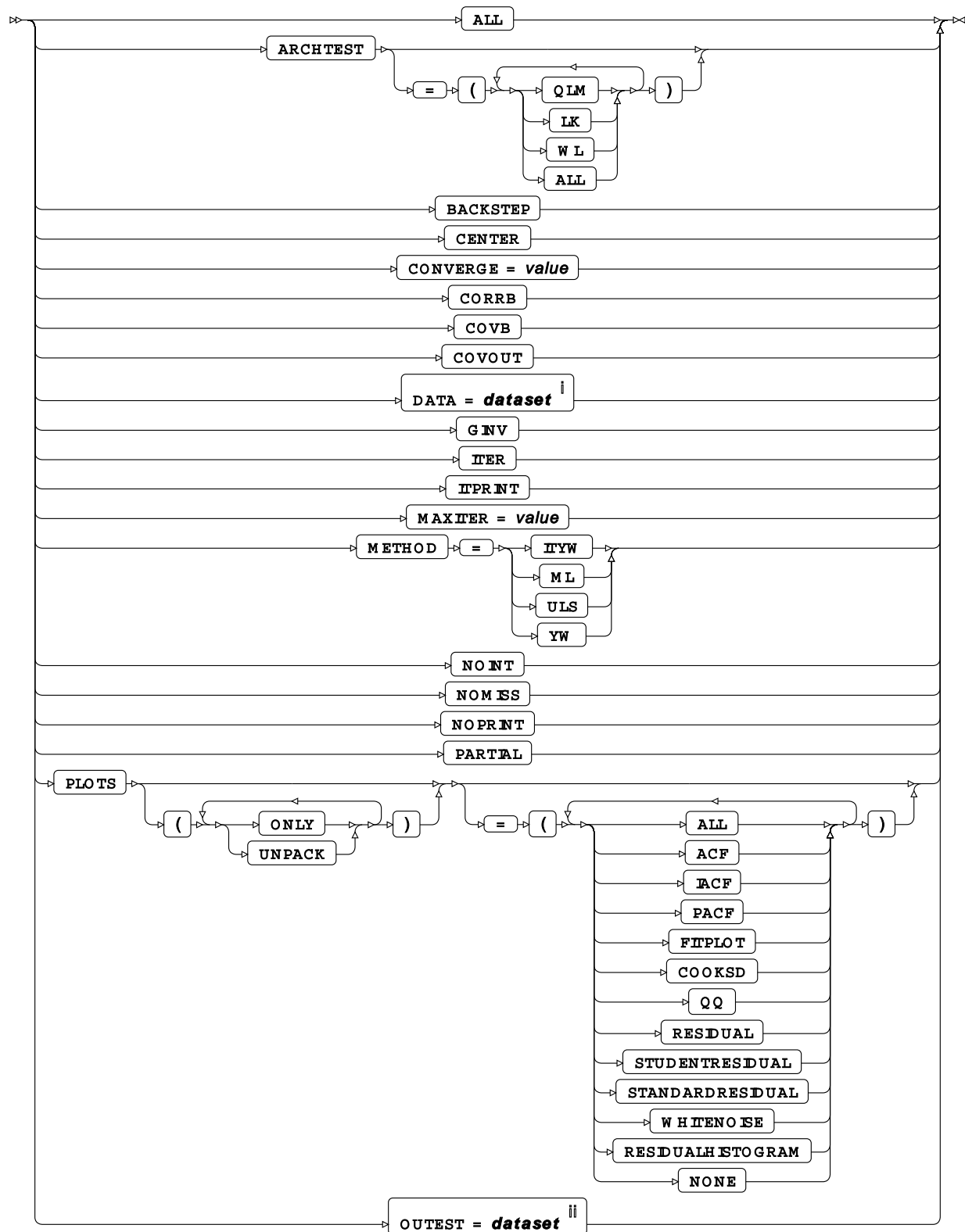
Supported statements

- *PROC AUTOREG* [↗](#) (page 3449)
- *ATTRIB* [↗](#) (page 3451)
- *BY* [↗](#) (page 3451)
- *FORMAT* [↗](#) (page 3451)
- *INFORMAT* [↗](#) (page 3451)
- *LABEL* [↗](#) (page 3452)
- *MODEL* [↗](#) (page 3452)
- *OUTPUT* [↗](#) (page 3453)
- *WHERE* [↗](#) (page 3454)

PROC AUTOREG



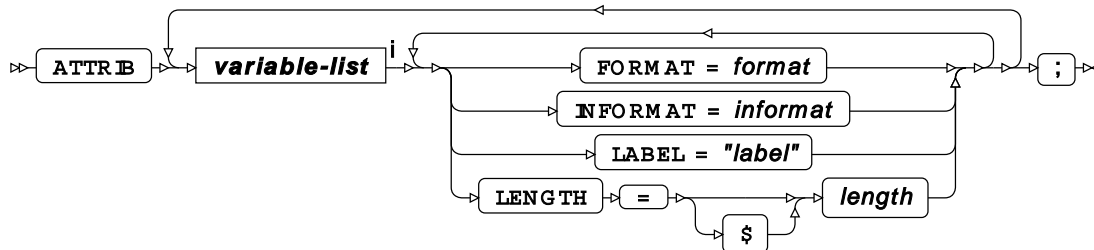
AUTOREG-options



ⁱ See *Input dataset* [↗](#) (page 16).

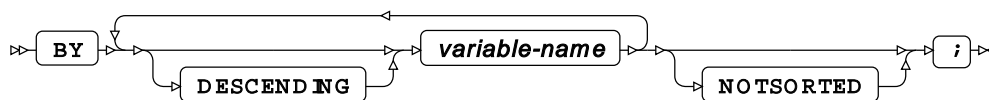
ⁱⁱ See *Output dataset* [↗](#) (page 16).

ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

BY



FORMAT



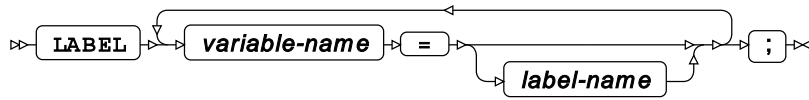
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

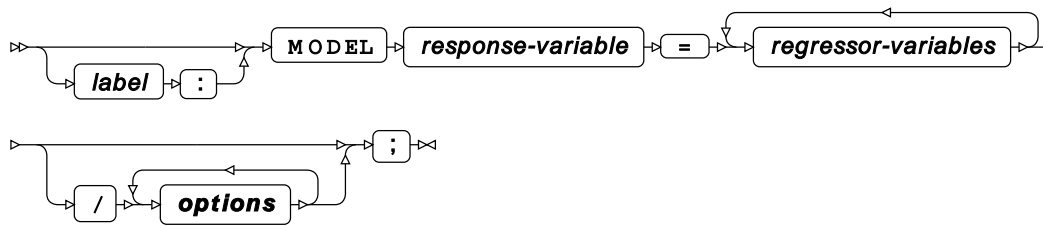


ⁱ See *Variable Lists* [↗](#) (page 32).

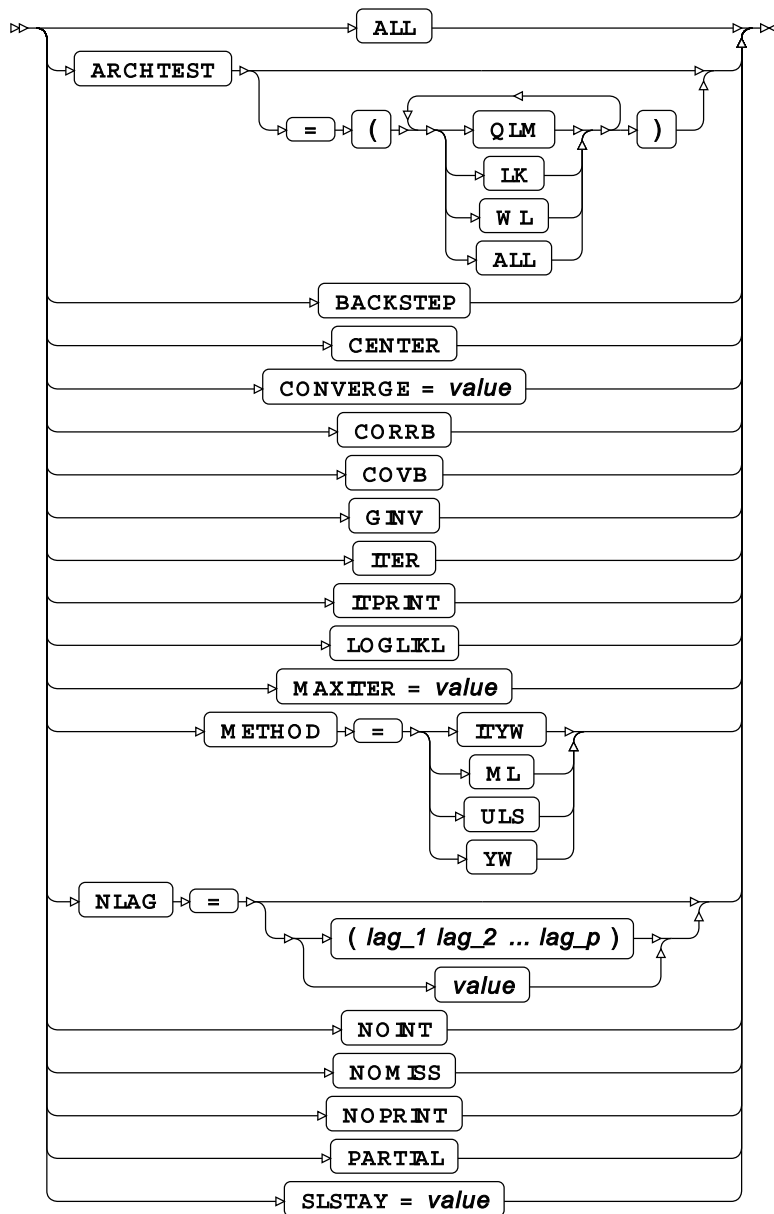
LABEL



MODEL



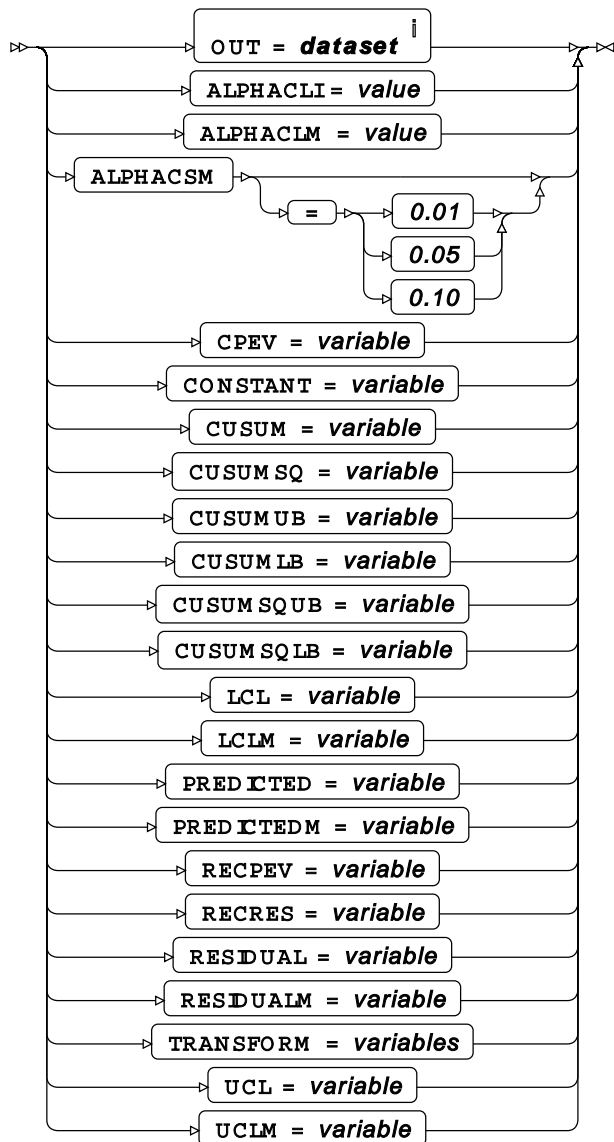
options



OUTPUT

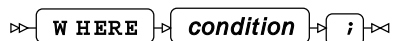


option



ⁱ See *Output dataset* [↗](#) (page 16).

WHERE



EXPAND procedure

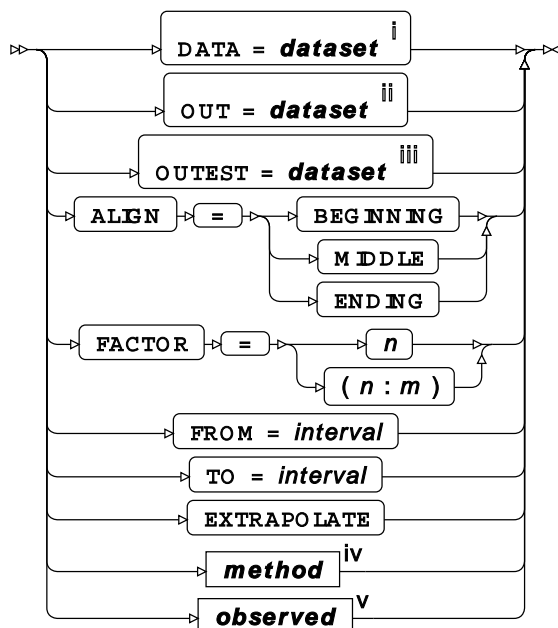
Supported statements

- *PROC EXPAND* [↗](#) (page 3455)
- *ATTRIB* [↗](#) (page 3456)
- *BY* [↗](#) (page 3456)
- *CONVERT* [↗](#) (page 3456)
- *FORMAT* [↗](#) (page 3460)
- *ID* [↗](#) (page 3461)
- *INFORMAT* [↗](#) (page 3461)
- *LABEL* [↗](#) (page 3461)
- *WHERE* [↗](#) (page 3461)

PROC EXPAND



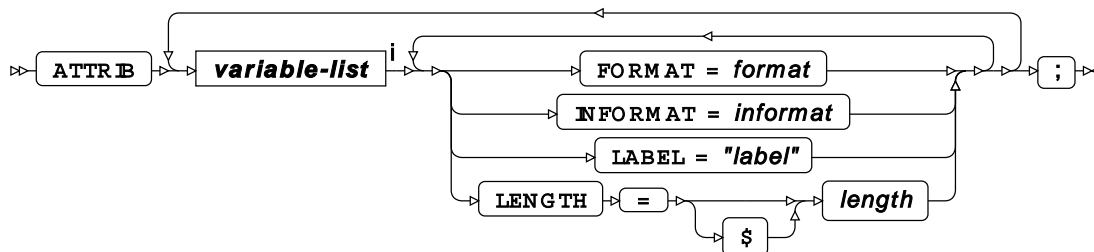
EXPAND-options



ⁱ See *Input dataset* [↗](#) (page 16).

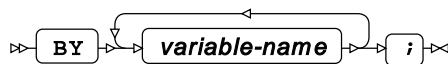
- ii See *Output dataset* [↗](#) (page 16).
- iii See *Output dataset* [↗](#) (page 16).
- iv See *Option METHOD* [↗](#) (page 3462).
- v See *Option OBSERVED* [↗](#) (page 3462).

ATTRIB

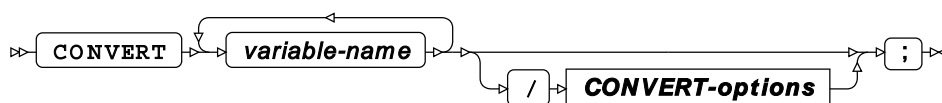


- ⁱ See *Variable Lists* [↗](#) (page 32).

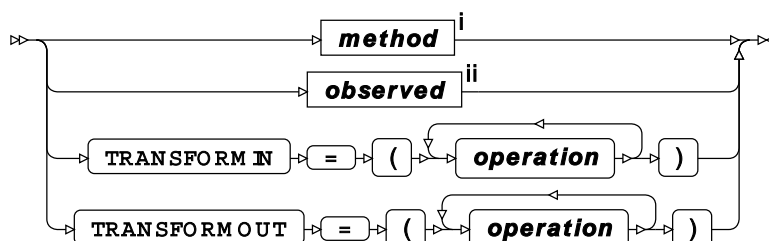
BY



CONVERT

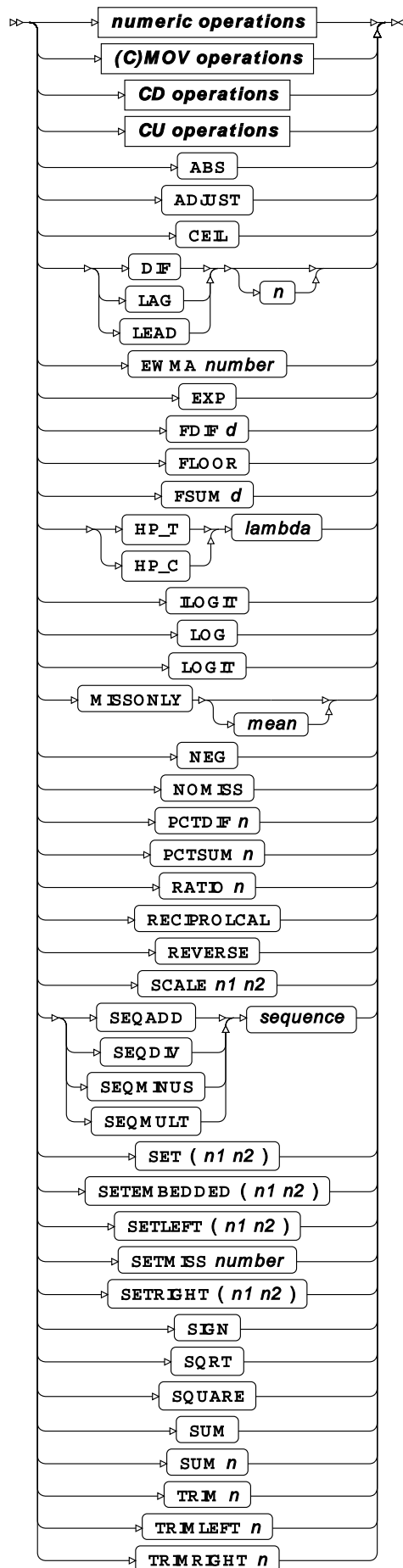


CONVERT-options

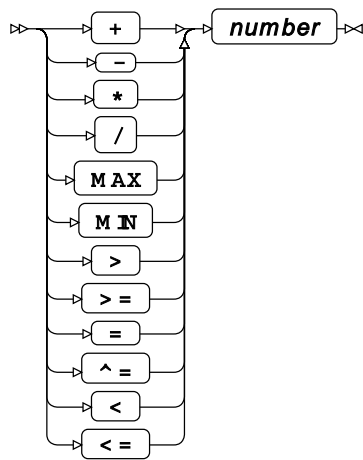


- ⁱ See *Option METHOD* [↗](#) (page 3462).
- ⁱⁱ See *Option OBSERVED* [↗](#) (page 3462).

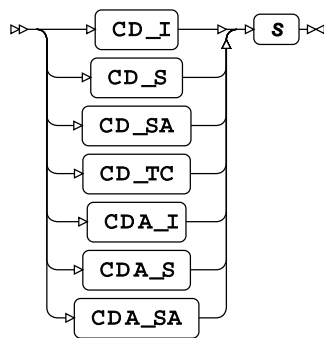
operation



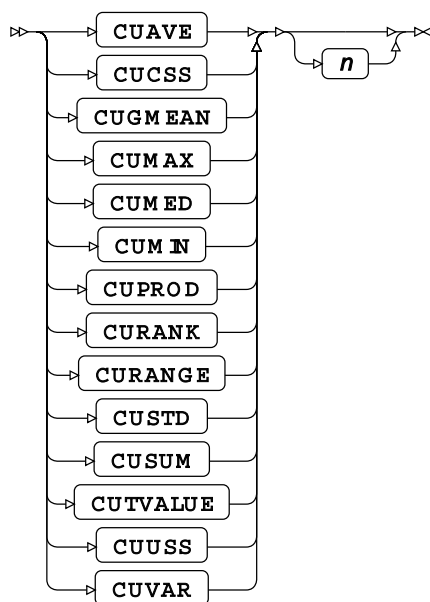
numeric operations



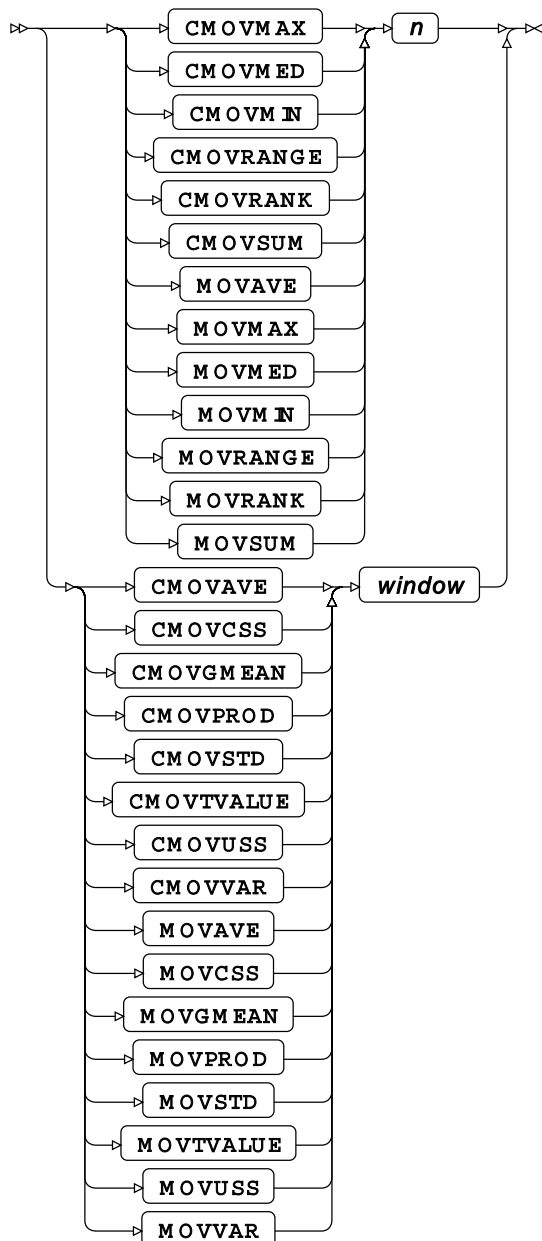
CD operations



CU operations



(C)MOV operations



FORMAT

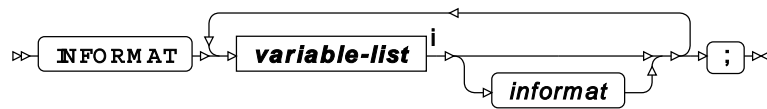


ⁱ See [Variable Lists](#) (page 32).

ID

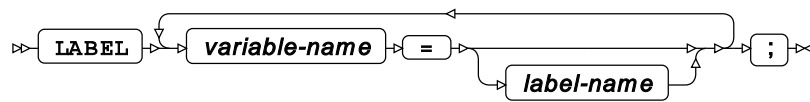


INFORMAT

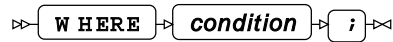


ⁱ See [Variable Lists](#) (page 32).

LABEL

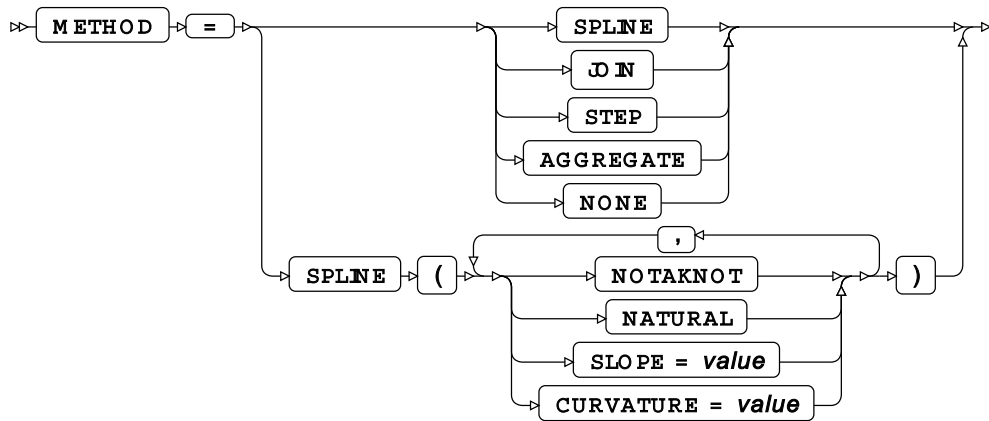


WHERE

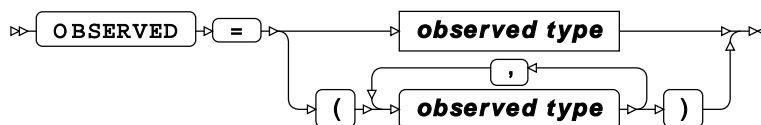


Common options

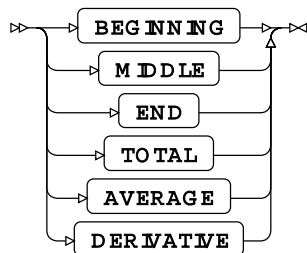
Option METHOD



Option OBSERVED



observed type



FORECAST procedure

Supported statements

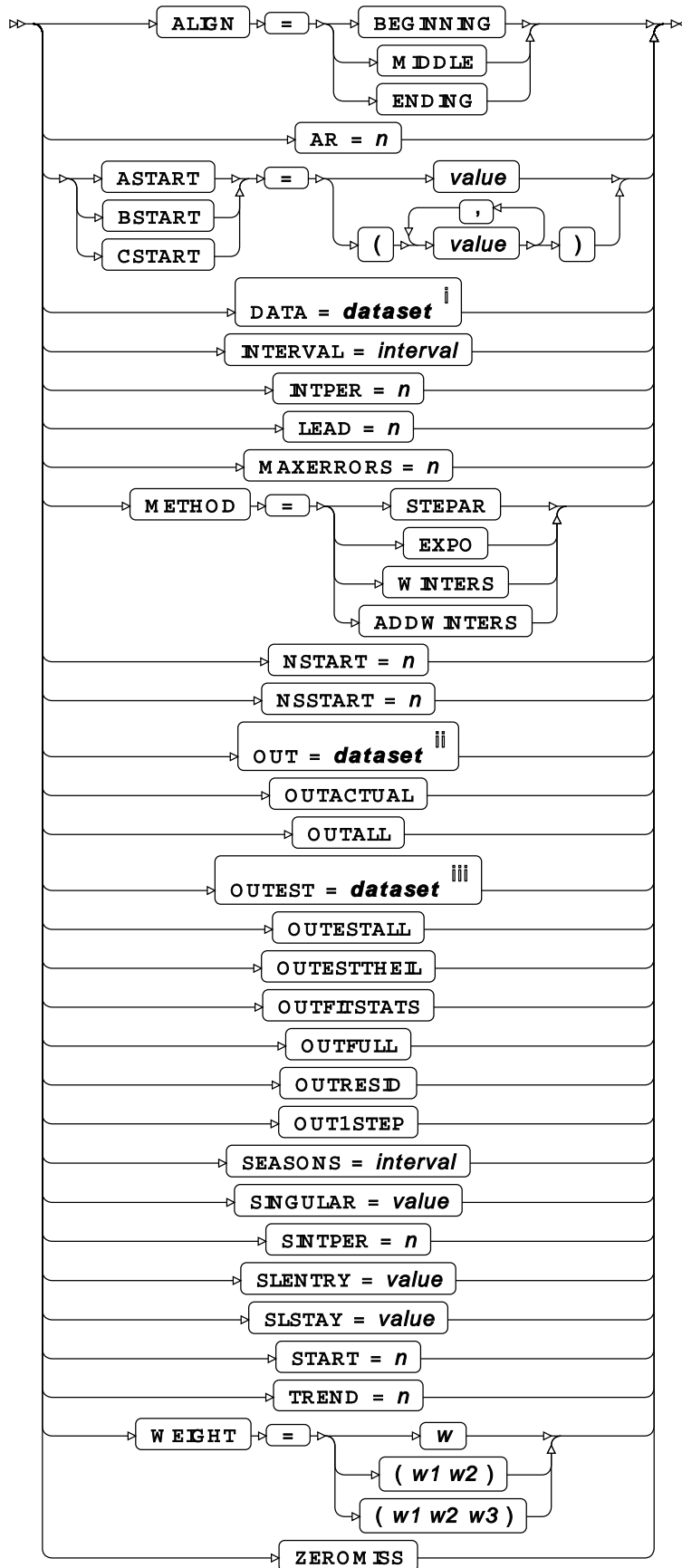
- *PROC FORECAST* [↗](#) (page 3463)
- *ATTRIB* [↗](#) (page 3465)

- *BY* [↗](#) (page 3465)
- *FORMAT* [↗](#) (page 3465)
- *ID* [↗](#) (page 3465)
- *INFORMAT* [↗](#) (page 3466)
- *LABEL* [↗](#) (page 3466)
- *VAR* [↗](#) (page 3466)
- *WHERE* [↗](#) (page 3466)

PROC FORECAST

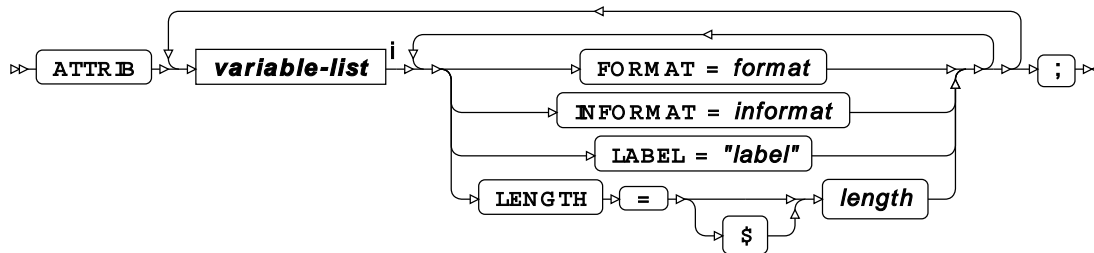


FORECAST-options



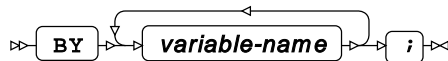
- ⁱ See *Input dataset* [↗](#) (page 16).
- ⁱⁱ See *Output dataset* [↗](#) (page 16).
- ⁱⁱⁱ See *Output dataset* [↗](#) (page 16).

ATTRIB

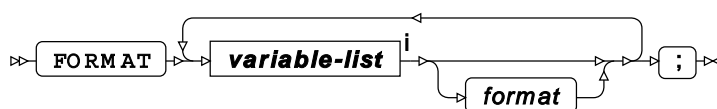


- ⁱ See *Variable Lists* [↗](#) (page 32).

BY

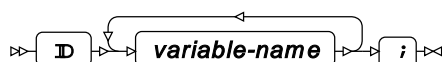


FORMAT



- ⁱ See *Variable Lists* [↗](#) (page 32).

ID

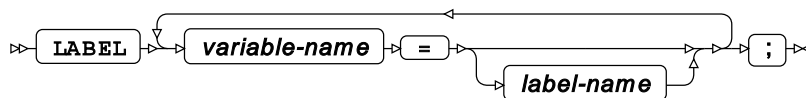


INFORMAT

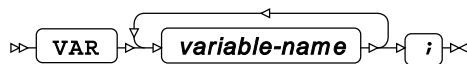


ⁱ See *Variable Lists* [↗](#) (page 32).

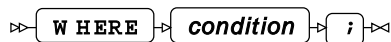
LABEL



VAR



WHERE



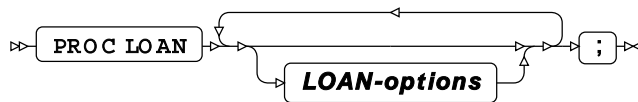
LOAN procedure

Supported statements

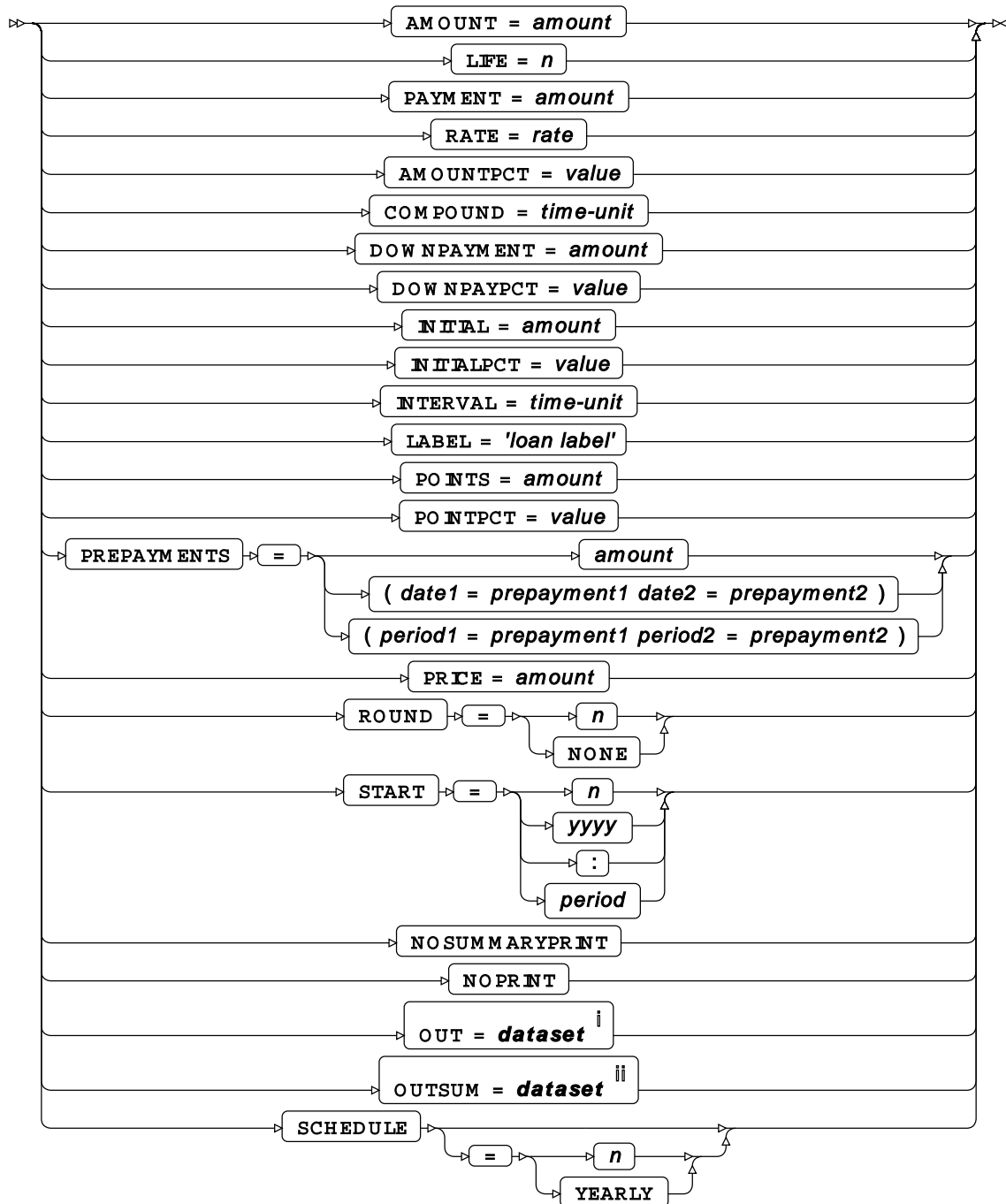
- *PROC LOAN* [↗](#) (page 3467)
- *ARM* [↗](#) (page 3469)
- *ATTRIB* [↗](#) (page 3469)
- *BALLOON* [↗](#) (page 3469)
- *BUYDOWN* [↗](#) (page 3470)
- *COMPARE* [↗](#) (page 3470)
- *FIXED* [↗](#) (page 3471)
- *FORMAT* [↗](#) (page 3471)

- *INFORMAT* [↗](#) (page 3471)
- *LABEL* [↗](#) (page 3471)
- *WHERE* [↗](#) (page 3471)

PROC LOAN



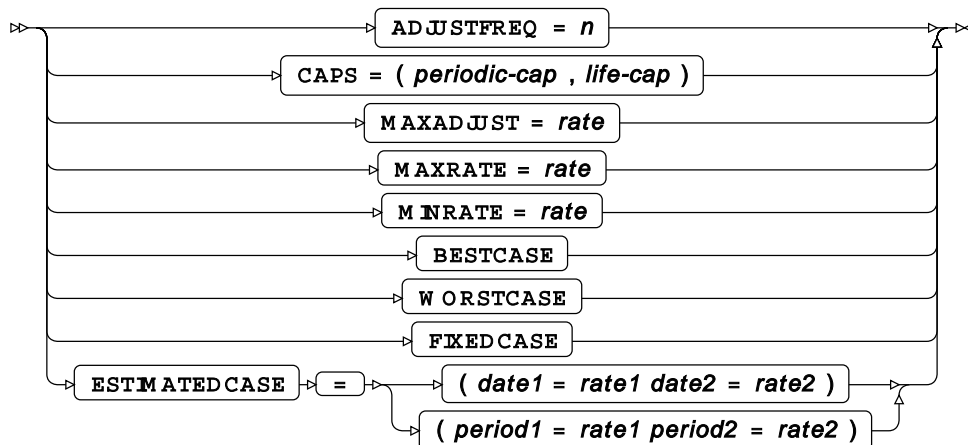
LOAN-options

ⁱ See *Output dataset* [\(page 16\)](#).ⁱⁱ See *Output dataset* [\(page 16\)](#).

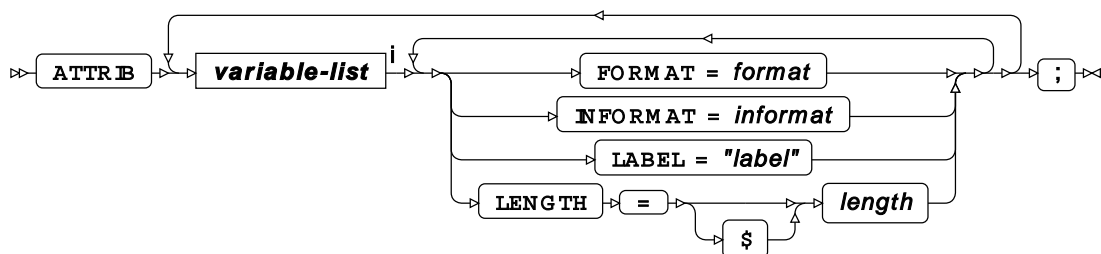
ARM



ARM-options



ATTRIB

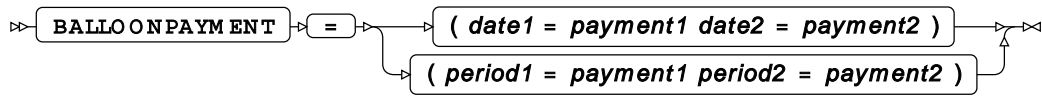


ⁱ See [Variable Lists](#) (page 32).

BALLOON



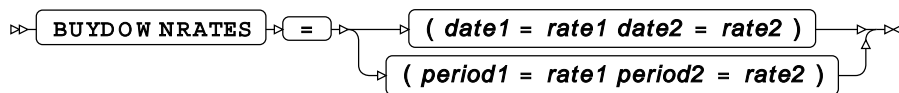
BALLOON-options



BUYDOWN



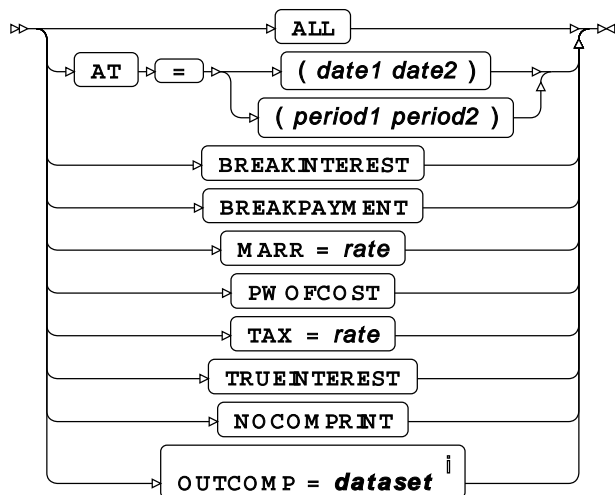
BUYDOWN-options



COMPARE

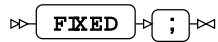


COMPARE-options



ⁱ See [Output dataset](#) (page 16).

FIXED



FORMAT



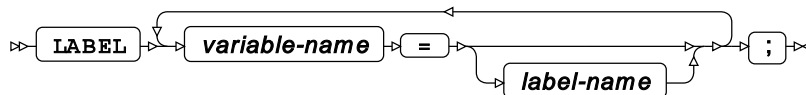
ⁱ See *Variable Lists* [↗](#) (page 32).

INFORMAT

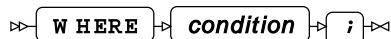


ⁱ See *Variable Lists* [↗](#) (page 32).

LABEL



WHERE

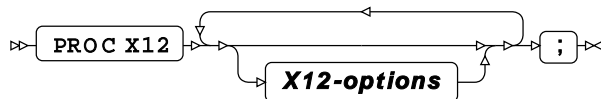


X12 procedure

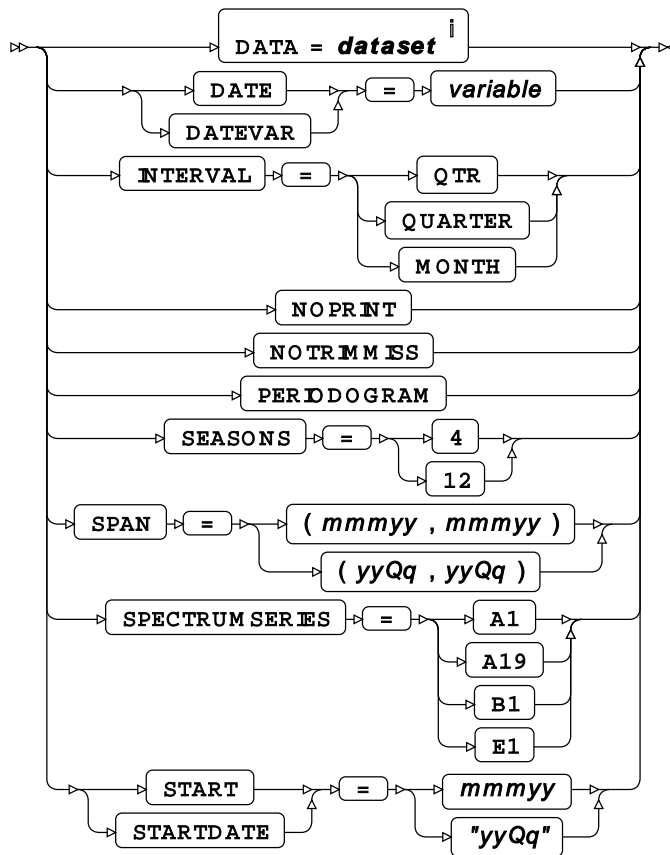
Supported statements

- *PROC X12* [↗](#) (page 3472)
- *ADJUST* [↗](#) (page 3473)
- *ARIMA* [↗](#) (page 3473)
- *ATTRIB* [↗](#) (page 3474)
- *BY* [↗](#) (page 3474)
- *ESTIMATE* [↗](#) (page 3474)
- *FORECAST* [↗](#) (page 3474)
- *FORMAT* [↗](#) (page 3475)
- *ID* [↗](#) (page 3475)
- *IDENTIFY* [↗](#) (page 3475)
- *INFORMAT* [↗](#) (page 3476)
- *LABEL* [↗](#) (page 3476)
- *OUTPUT* [↗](#) (page 3476)
- *REGRESSION* [↗](#) (page 3478)
- *TABLES* [↗](#) (page 3478)
- *TRANSFORM* [↗](#) (page 3480)
- *VAR* [↗](#) (page 3480)
- *X11* [↗](#) (page 3480)
- *WHERE* [↗](#) (page 3480)

PROC X12

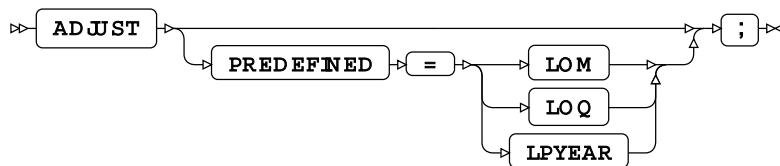


X12-options

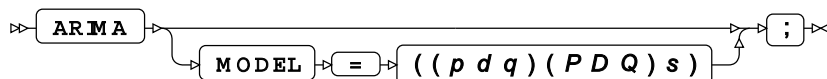


ⁱ See *Input dataset* [↗](#) (page 16).

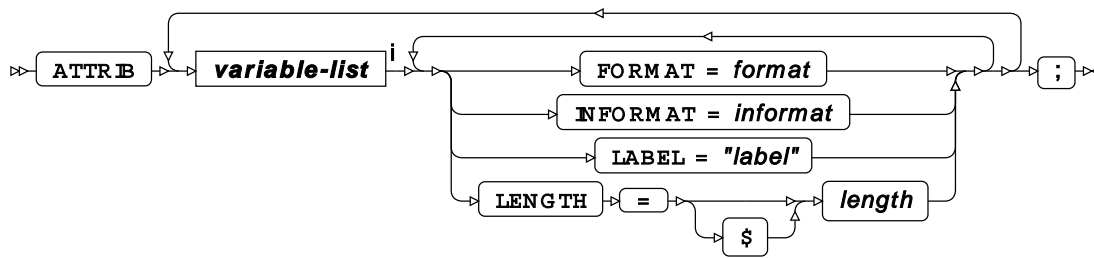
ADJUST



ARIMA

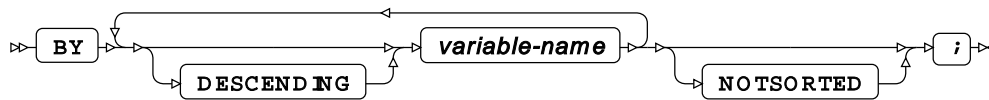


ATTRIB



ⁱ See *Variable Lists* [↗](#) (page 32).

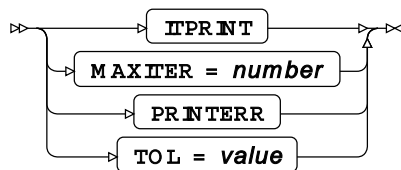
BY



ESTIMATE



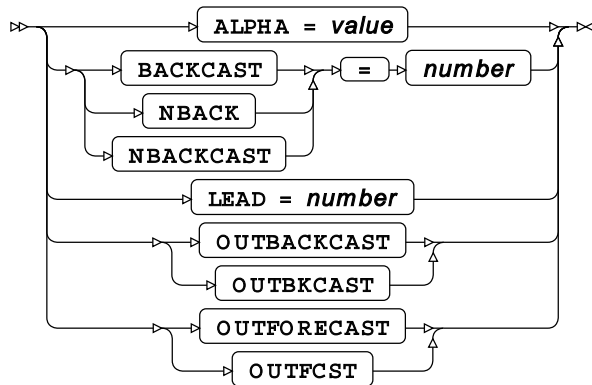
ESTIMATE-options



FORECAST



FORECAST-options

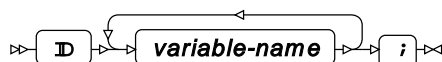


FORMAT



ⁱ See *Variable Lists* [↗](#) (page 32).

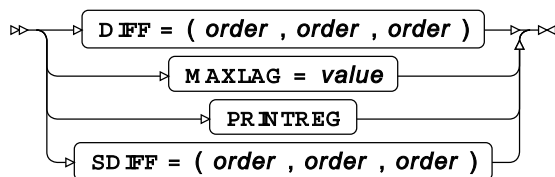
ID



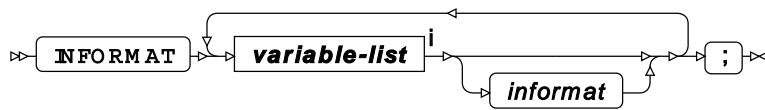
IDENTIFY



IDENTIFY-options

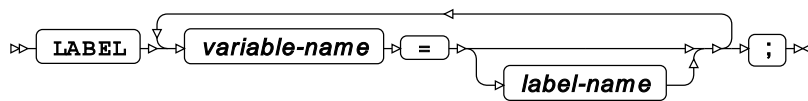


INFORMAT

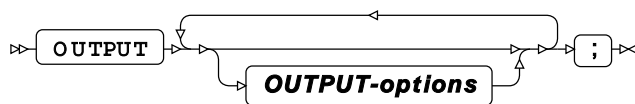


ⁱ See *Variable Lists* [↗](#) (page 32).

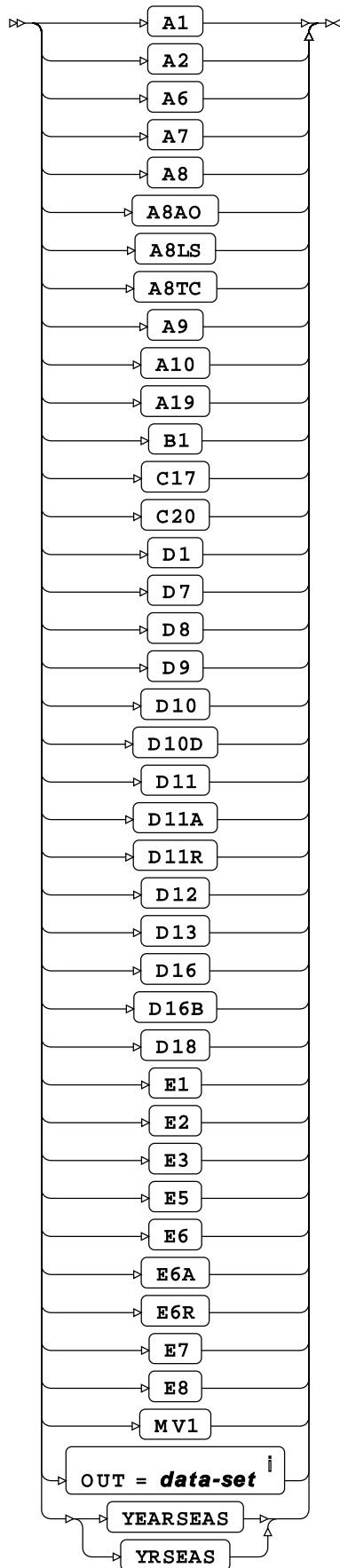
LABEL



OUTPUT

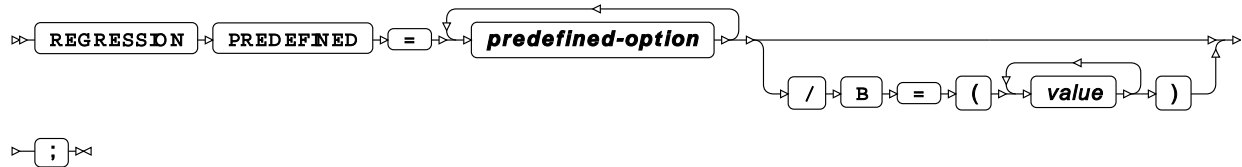


OUTPUT-options

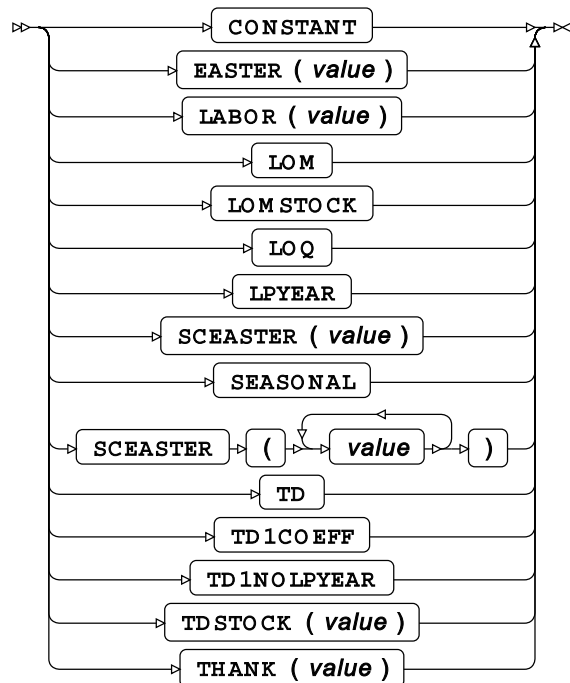


ⁱ See *Output dataset* [↗](#) (page 16).

REGRESSION



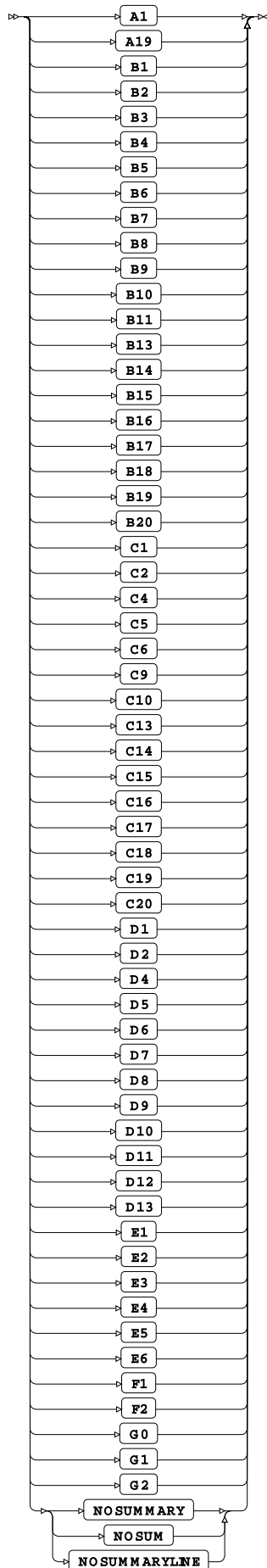
predefined-option



TABLES



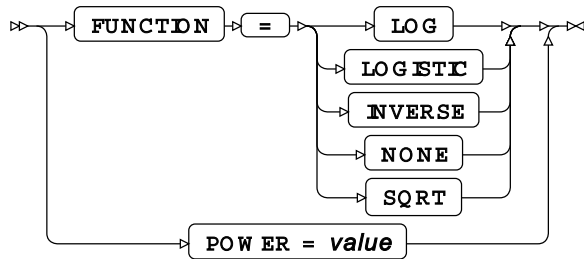
TABLES-option



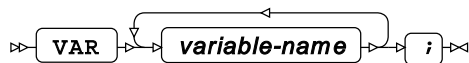
TRANSFORM



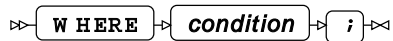
TRANSFORM-option



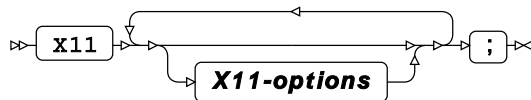
VAR



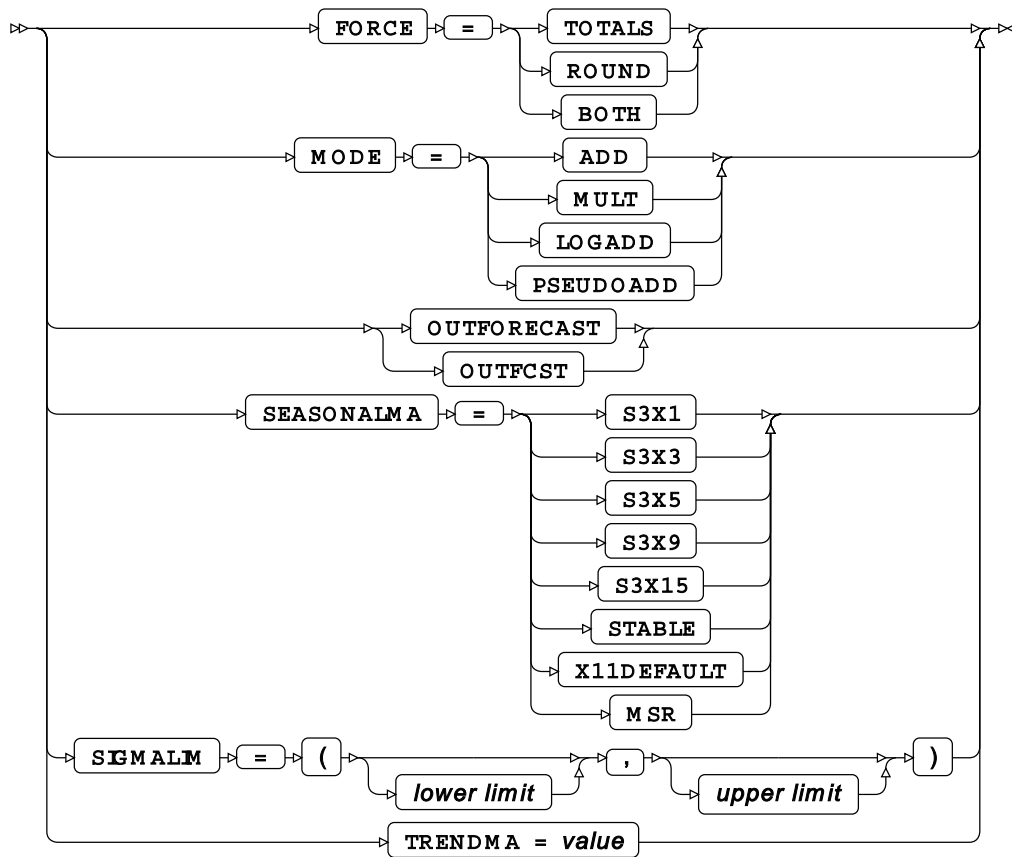
WHERE



X11



X11-options



WPS Communicate

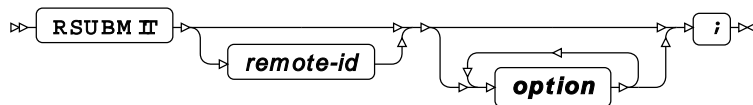
Global statements

ENDRSUBMIT

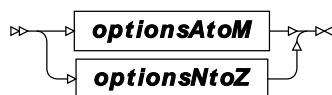


This statement indicates the end of a block of code that began with an `RSUBMIT` statement.

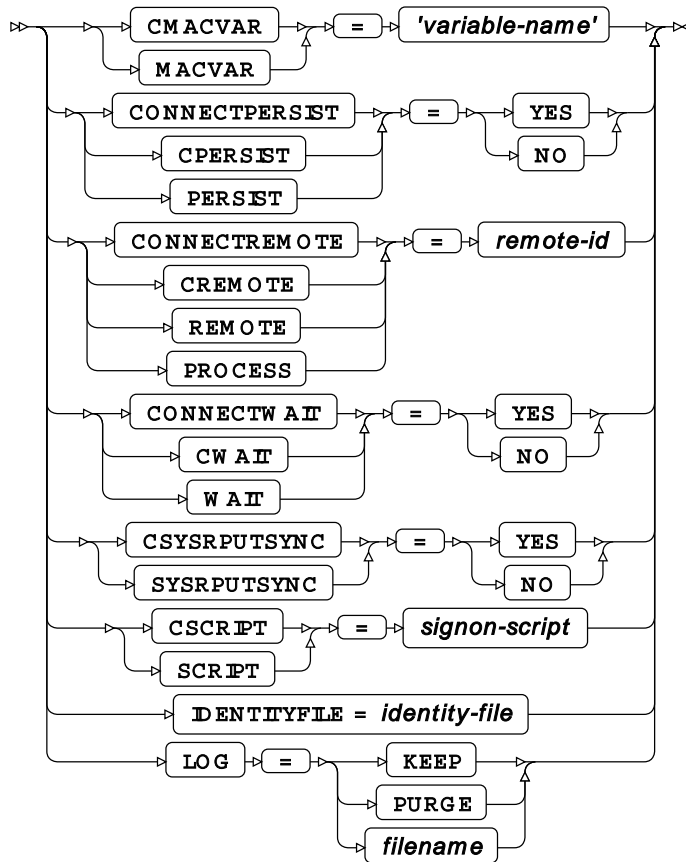
RSUBMIT

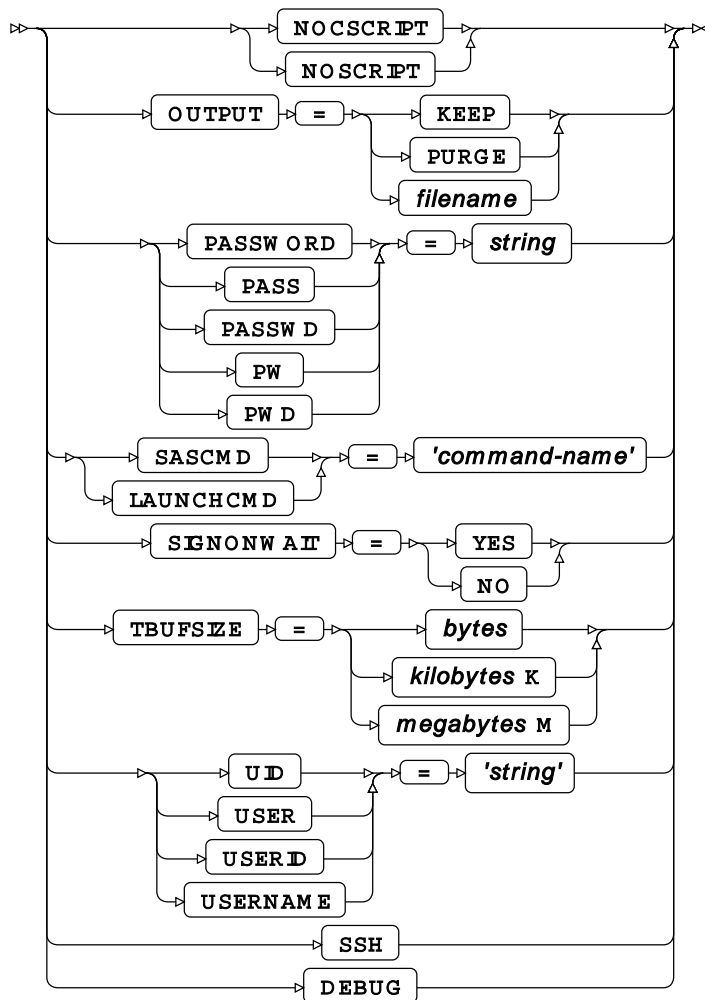


option



options A to M



options **N** to **Z**

This statement marks the beginning of a block of program code to be submitted to a (usually remote) host for execution.

CMACVAR, MACVAR

This option specifies a macro variable whose value is bound to the completion status of the current `RSUBMIT` block.

CONNECTPERSIST, CPERSIST, PERSIST

This option signifies whether or not an automatic signoff occurs after a `SIGNON` and `RSUBMIT`.

CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option identifies the remote machine to which a connection will be established, either directly or by naming a macro variable that contains the address.

Note:

If the `CONNECTREMOTE` option is used with the name of the remote host specifically provided as a macro variable, then no ampersand should be placed before the macro variable name. The correct syntax is illustrated in the following fragment:

```
...
%LET HostName = RemoteHost;

options ssh_hostvalidation=none;
signon connectremote=HostName ssh /* Not &HostName */
user = <username>
password = <password>
launchcmd = '<location-of-wps-executable> -dmr';
...
```

CONNECTWAIT, CWAIT, WAIT

This option determines if the `RSUBMIT` block is to be run in asynchronous or synchronous mode, by setting it to `NO` or `YES` respectively.

CSYSRPUTSYNC, SYSRPUTSYNC

If set to `YES`, this option forces macro variables to be defined when `%SYSRPUT` executes.

CSCSCRIPT, SCRIPT

This option identifies a signon script.

IDENTITYFILE

This option specifies a file containing authentication information, such as SSH keys.

NOSCRIPT, NOCSCRIPT

This option indicates that no script should be used to sign on.

LOG

This option defines whether the system log should be kept, purged or sent to a specific file.

OUTPUT

This option defines whether the output of the sub-program should be kept, purged or sent to a specific file.

PASSWORD, PASS, PASSWD, PW, PWD

This option is used to specify a password for remote authorisation.

SASCMD, LAUNCHCMD

When present, this option is used to specify the command required to launch WPS on the remote machine.

SIGNONWAIT

This option stipulates that a `SIGNON` should finish before permitting subsequent processing.

TBUFSIZE

This option specifies the WPS COMMUNICATE message buffer size.

UID, USER, USERID, USERNAME

When present, this option specifies the user name.

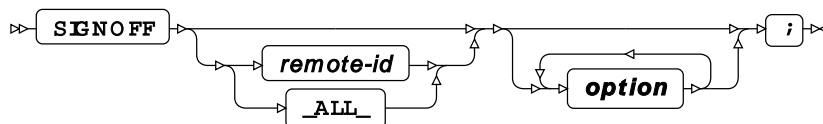
SSH

This option specifies that the connection will utilise the encrypted SSH protocol.

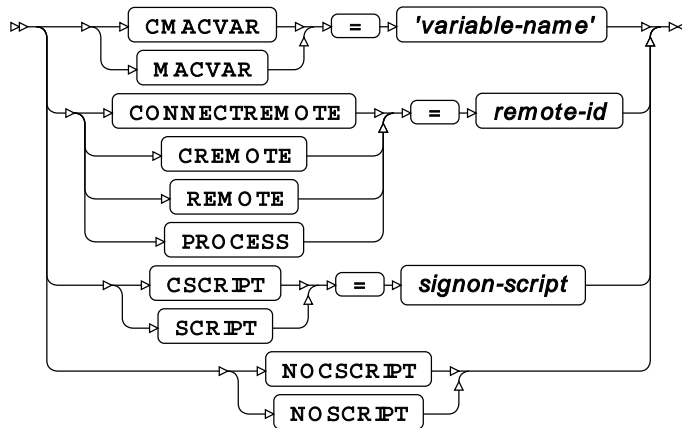
DEBUG

This option specifies that extra debugging messages are written to the sytem log.

SIGNOFF



option



This statement closes down a connection with a remote server, following the execution of a remotely executed block of code.

CMACVAR, MACVAR

This option specifies a macro variable associated with the remote session and whose value is bound to the completion status of the current **SIGNOFF** statement.

CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option names the remote session from which you wish to sign off.

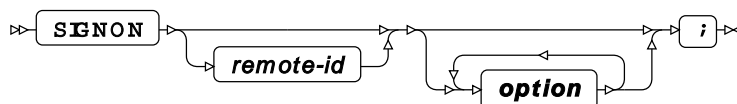
CSCRIPT, SCRIPT

This option identifies a script to be executed during signoff.

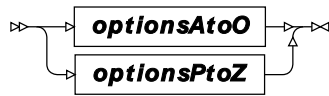
NOSCRIPT, NOCSCRIPT

This option indicates that no script should be involved in the signoff process.

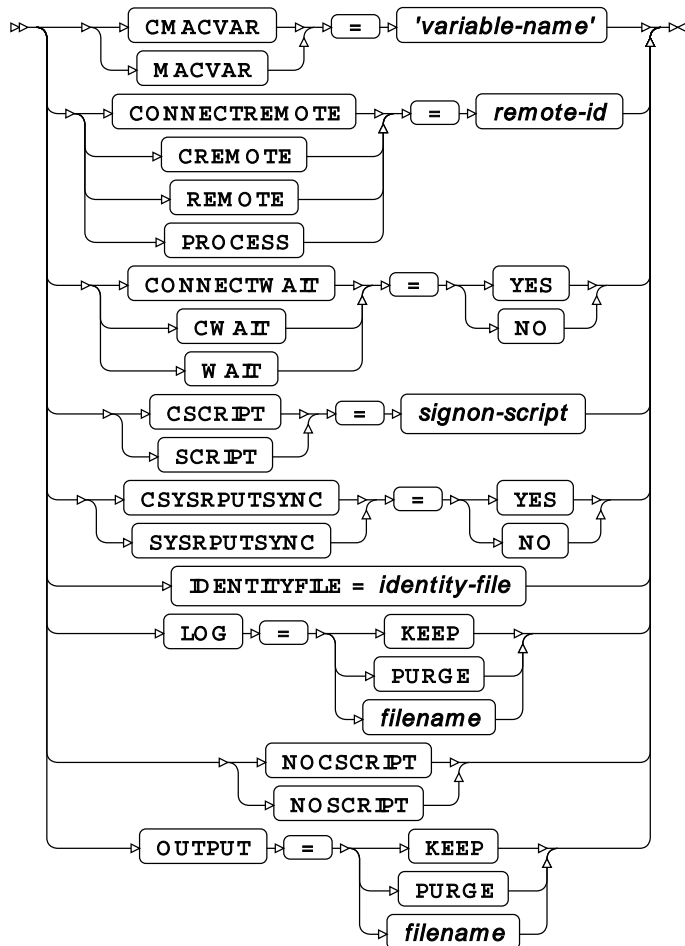
SIGNON



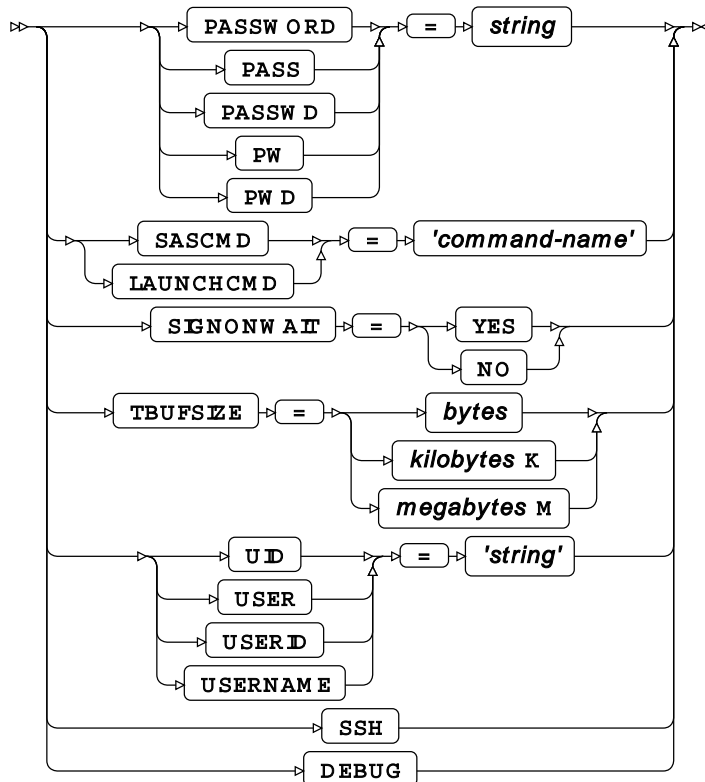
option



options A to o



options P to Z



This statement and its options provide the information necessary to specify where the remote WPS installation is located, plus credentials to connect and log in to the remote server, prior to invoking a block of remotely executed code.

CMACVAR, MACVAR

This option specifies a macro variable associated with the remote session and whose value is bound to the completion status of the current `SIGNON` statement.

CONNECTREMOTE, CREMOTE, REMOTE, PROCESS

This option names the remote session.

Note that if the `CONNECTREMOTE` option is used with the name of the remote host specifically provided as a macro variable, then (perhaps counterintuitively) no ampersand should be placed before the macro variable name. The correct syntax is illustrated in the following fragment:

```
...
%LET HostName = RemoteHost;

options ssh_hostvalidation=none;
signon connectremote=HostName ssh /* Not &HostName */
user = <username>
password = <password>
launchcmd = '<location-of-wps-executable> -dmr';
...
```

CONNECTWAIT, CWAIT, WAIT

This option determines if the `RSUBMIT` block is to be run in asynchronous or synchronous mode, by setting it to `NO` or `YES` respectively.

CSYSRPUTSYNC, SYSRPUTSYNC

If set to `YES`, this option forces macro variables to be defined when `%SYSRPUT` executes.

CSCSCRIPT, SCRIPT

This option identifies a signon script.

IDENTITYFILE

This option specifies a file containing authentication information, such as SSH keys.

NOSCRIPT, NOCSCRIPT

This option indicates that no script should be used to sign on.

LOG

This option defines whether the system log should be kept, purged or sent to a specific file.

OUTPUT

This option defines whether the output of the sub-program should be kept, purged or sent to a specific file.

PASSWORD, PASS, PASSWD, PW, PWD

This option is used to specify a password for remote authorisation.

SASCMD, LAUNCHCMD

When present, this option is used to specify the command required to launch WPS on the remote machine.

SIGNONWAIT

This option stipulates that a `SIGNON` should finish before permitting subsequent processing.

TBUFSIZE

This option specifies the WPS COMMUNICATE message buffer size.

UID, USER, USERID, USERNAME

When present, this option specifies the user name.

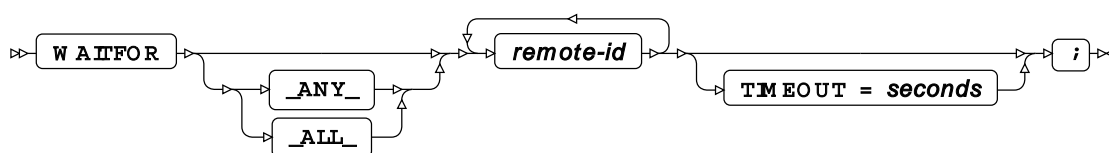
SSH

This option specifies that the connection will utilise the encrypted SSH protocol.

DEBUG

This option specifies that extra debugging messages are written to the sytem log.

WAITFOR



In that the above diagram applies to WPS Communicate only, the `WAITFOR _ALL_` statement suspends execution of the current session until processing is complete for **all** of the server `remote-ids`, or until the `TIMEOUT` interval, if specified, has expired.

If you use `WAITFOR _ANY_`, or simply `WAITFOR`, instead of `WAITFOR _ALL_`, then execution of the session will only be suspended until processing is complete on one of the server `remote-ids` (or until the `TIMEOUT` interval, if specified, has expired).

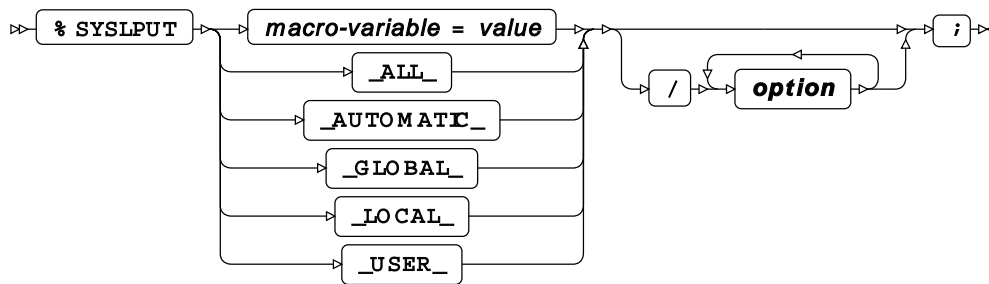
Note:

As implied above, the default is `_ANY_` rather than `_ALL_` if no argument is supplied between `WAITFOR` and the `remote-ids`.

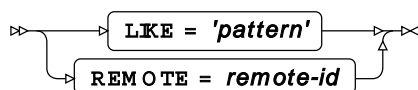
Macro processor statements

These statements enable you to create and retrieve the value of a macro variable on a remote server.

%SYSLPUT



option



This statement creates a macro variable on a remote host with which you have established a WPS Communicate session. It should be placed outside of the corresponding `RSUBMIT` block.

%SYSRPUT



This statement retrieves a macro variable from a remote host to which there is an established WPS Communicate session, creating an identical local macro variable. It should be placed inside the corresponding `RSUBMIT` block.

WPS Communicate procedures

These procedures enable you to transfer files, libraries or datasets to and from a remote host.

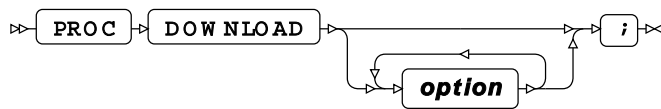
DOWNLOAD Procedure

This procedure downloads one or more files, libraries or datasets from a remote host. It can only be invoked from inside an `RSUBMIT` block.

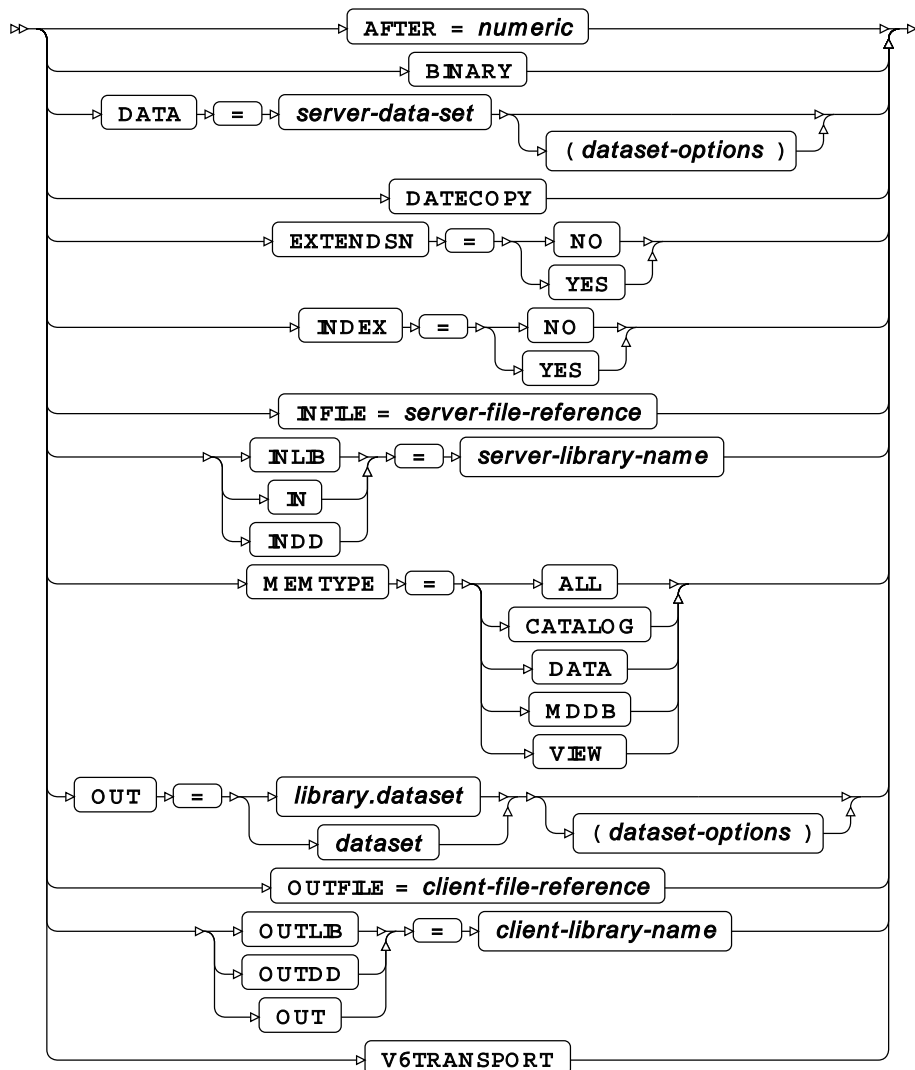
Supported statements

- `PROC DOWNLOAD` [↗](#) (page 3493)
- `EXCLUDE` [↗](#) (page 3496)
- `SELECT` [↗](#) (page 3496)
- `WHERE` [↗](#) (page 3497)

PROC DOWNLOAD



option



AFTER

Specifies a numeric modification date, ensuring that only datasets or libraries modified after this date are downloaded. This option is invalid for external file downloads.

BINARY

Valid only when downloading external files, this option specifies that the transfer should be an exact, binary copy.

DATA

Specifies the name of a dataset to be downloaded.

DATECOPY

When present, this option indicates that a remote dataset's creation date and time should be retained when it is downloaded. This option is invalid for external file downloads.

EXTENDSN

Specifies if short numeric variables should have their lengths extended. This option is invalid for external file downloads and might be considered if transferring datasets from a mainframe to a PC.

INDEX

For remote datasets that have indexes, this indicates whether these indexes should be re-established on the local machine after the download. This option is invalid for external files downloads.

INFILE

Specifies the name of a remote external file to download. If this option is present, so must the `OUTFILE=` option be.

INLIB

Specifies the name of the remote library. This option is invalid for external file downloads.

OUT

Specifies the name of the receiving local dataset. This option is invalid for external file downloads.

OUTFILE

Specifies the name of local file to receive an external file download. If this option is present, so must the `INFILE` option be.

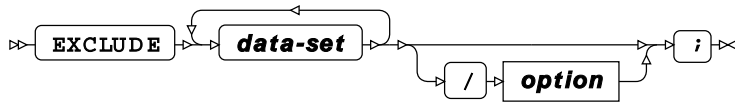
OUTLIB

Specifies the name of the local library into which a remote dataset is downloaded. This option is invalid for external file downloads.

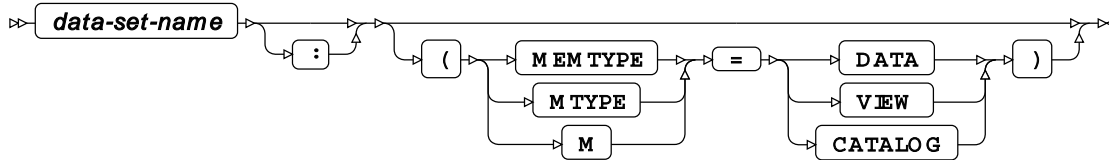
V6TRANSPORT

This is a translation option when exchanging data between two different versions.

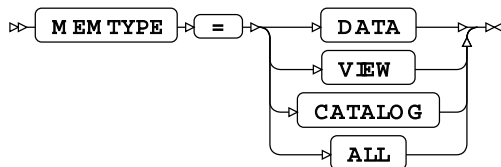
EXCLUDE



data-set



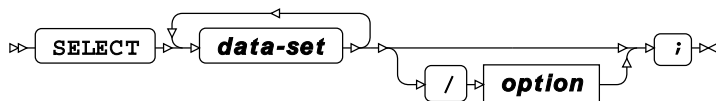
option



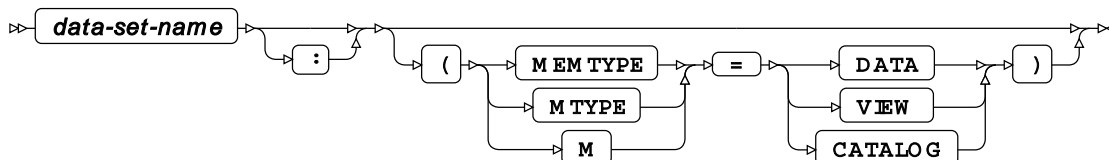
MEMTYPE

This option specifies the member types to be downloaded - see the syntax diagram above. This option is invalid for external file downloads.

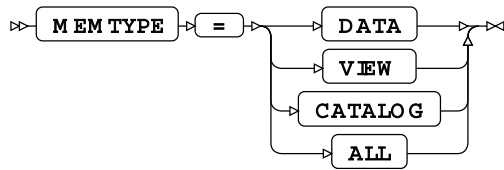
SELECT



data-set



option



MEMTYPE

See statement `EXCLUDE`.

WHERE

Restricts the observations to be processed.



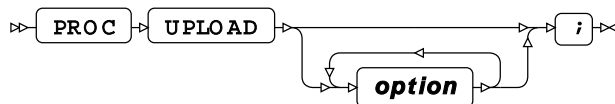
UPLOAD Procedure

This procedure uploads one or more files, libraries or datasets to a remote host. It can only be invoked from inside an `RSUBMIT` block.

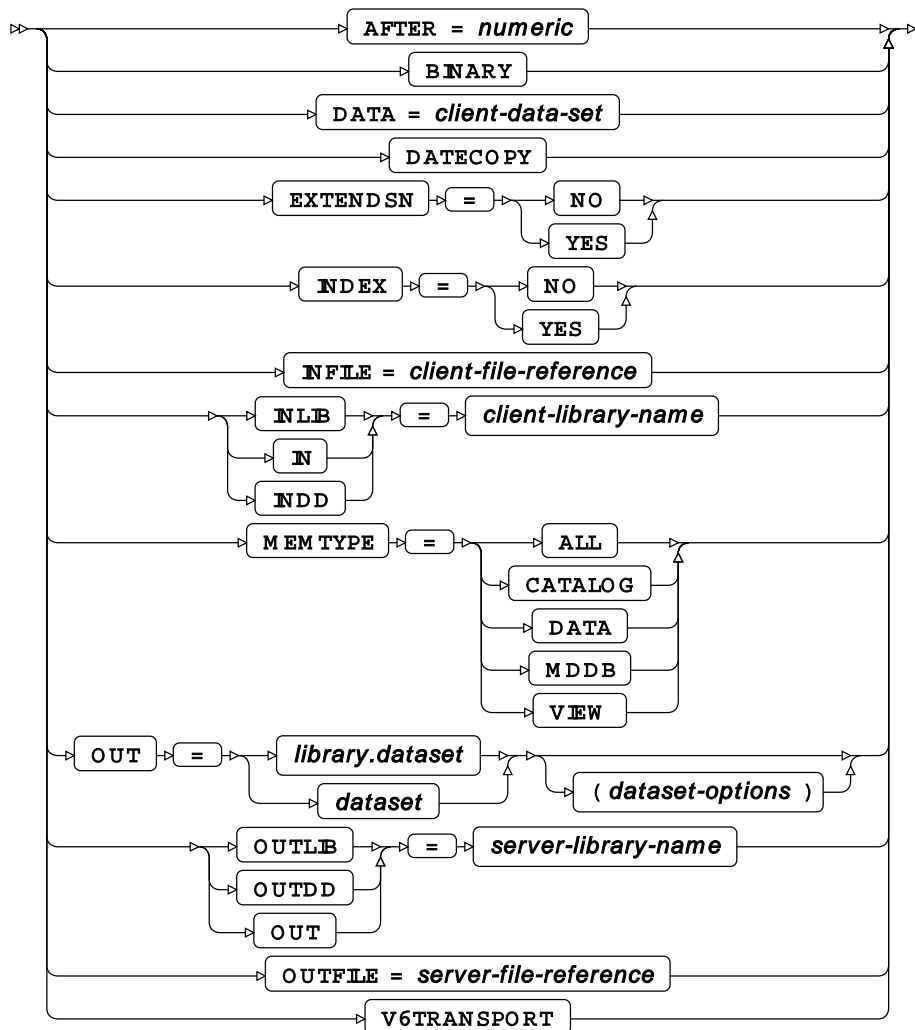
Supported statements

- `PROC UPLOAD` [↗](#) (page 3497)
- `EXCLUDE` [↗](#) (page 3500)
- `SELECT` [↗](#) (page 3500)
- `WHERE` [↗](#) (page 3501)

PROC UPLOAD



option

**AFTER**

Specifies a numeric modification date, ensuring that only datasets or libraries modified after this date are uploaded. This option is invalid for external file uploads.

BINARY

Valid only when uploading external files, this option specifies that the transfer should be an exact, binary copy.

DATA

Specifies the name of a dataset to be uploaded.

DATECOPY

When present, this option indicates that a local dataset's creation date and time should be retained when it is uploaded. This option is invalid for external file uploads.

EXTENDSN

Specifies if short numeric variables should have their lengths extended. This option is invalid for external file uploads and might be considered if transferring datasets to a mainframe from a PC.

INDEX

For local datasets that have indexes, this indicates whether these indexes should be re-established on the remote machine after the upload. This option is invalid for external files uploads.

INFILE

Specifies the name of a local external file to upload. If this option is present, so must the `OUTFILE=` option be.

INLIB

Specifies the name of the local library. This option is invalid for external file uploads.

MEMTYPE

This option specifies the member types to be uploaded - see the syntax diagram above. This option is invalid for external file uploads.

OUT

Specifies the name of the receiving remote dataset. This option is invalid for external file uploads.

OUTFILE

Specifies the name of remote file to receive an external file upload. If this option is present, so must the `INFILE` option be.

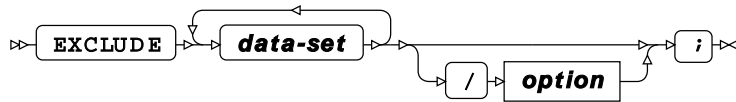
OUTLIB

Specifies the name of the remote library into which a local dataset is uploaded. This option is invalid for external file uploads.

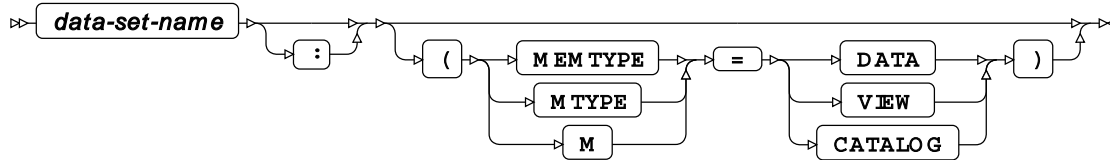
V6TRANSPORT

This is a translation option when exchanging data between two different versions.

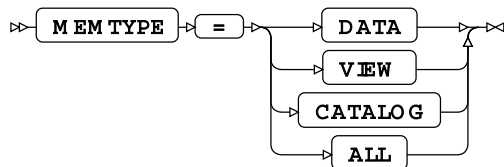
EXCLUDE



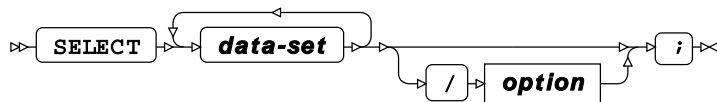
data-set



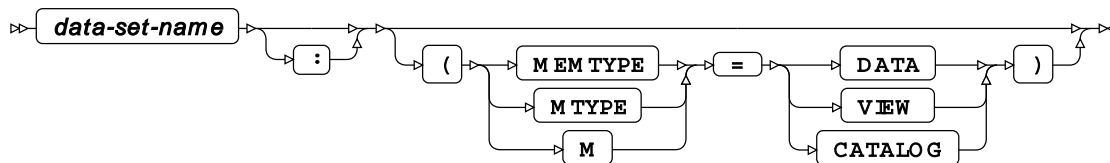
option



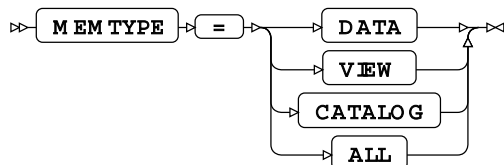
SELECT



data-set



option



WHERE

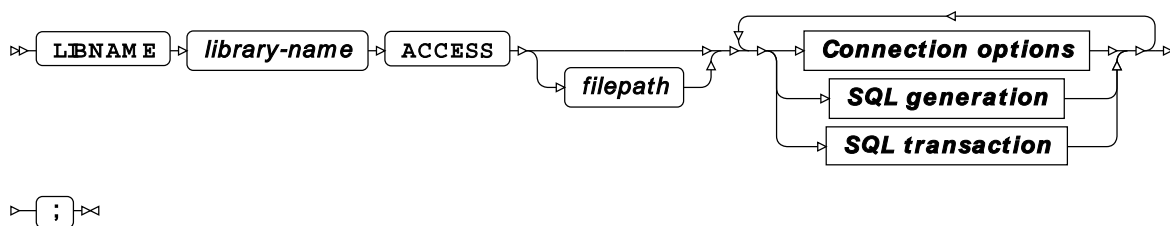
Restricts the observations to be processed.



Data Engines

WPS Engine for Access

ACCESS



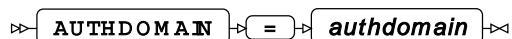
;

Connection options

ACCESS

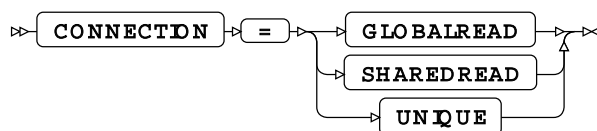


AUTHDOMAIN



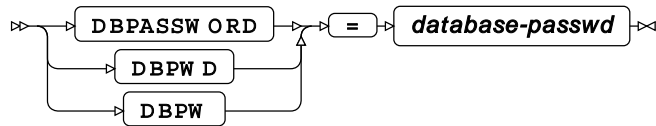
Type: String

CONNECTION



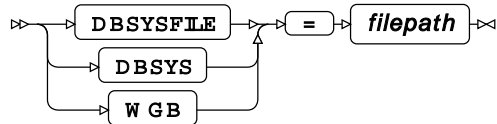
Default value: SHAREDREAD

DBPASSWORD



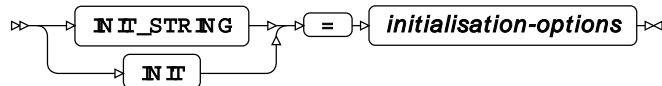
Type: String

DBSYSFILE



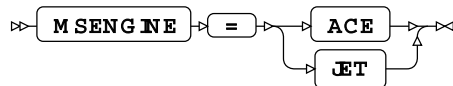
Type: String

INIT_STRING

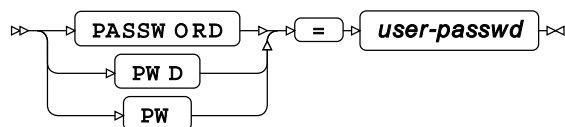


Type: String

MSENGINE

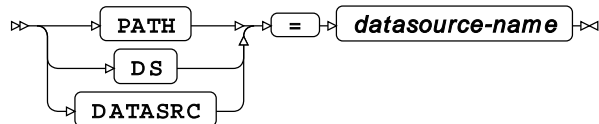


PASSWORD



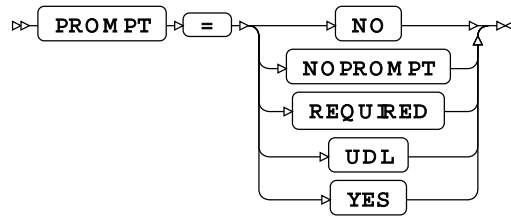
Type: String

PATH

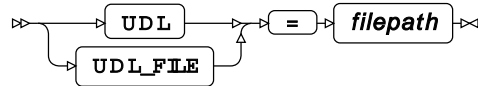


Type: String

PROMPT

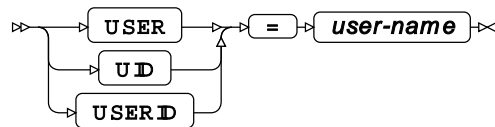


UDL



Type: String

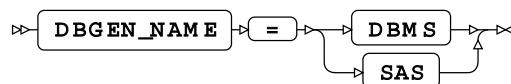
USER



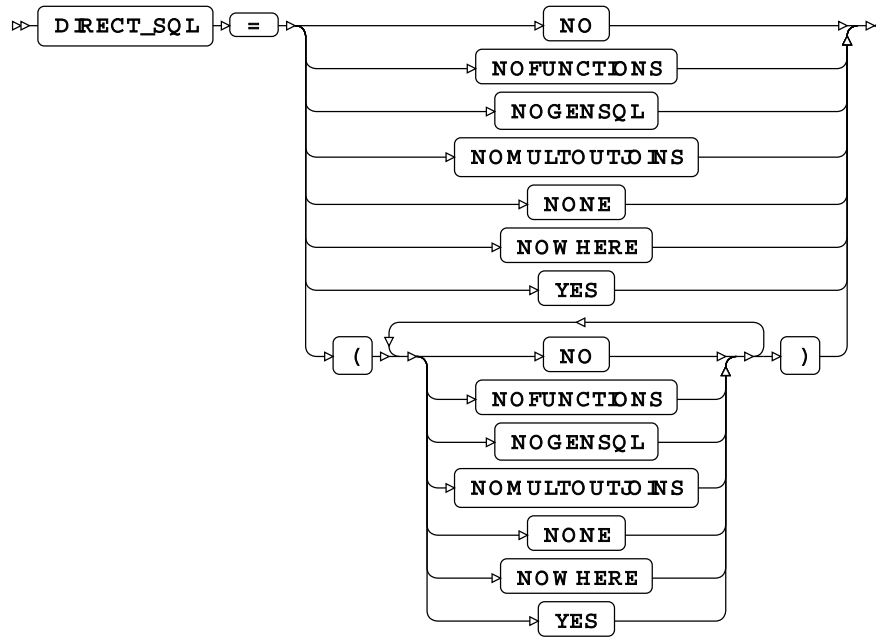
Type: String

SQL generation

DBGEN_NAME



DIRECT_SQL

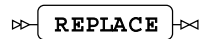


LABEL



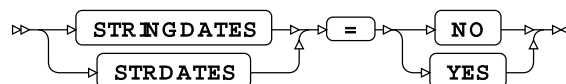
Type: Keyword

REPLACE

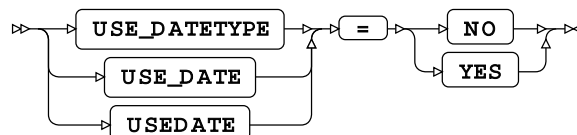


Type: Keyword

STRINGDATES

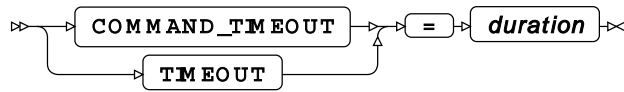


USE_DATATYPE



SQL transaction

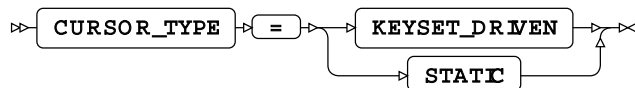
COMMAND_TIMEOUT



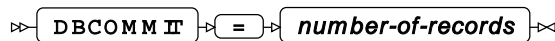
Type: Numeric

Minimum value: 0

CURSOR_TYPE



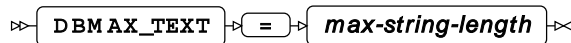
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

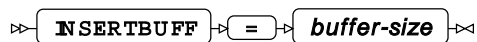


Type: Numeric

Minimum value: 1

Maximum value: 32767

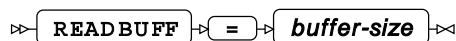
INSERTBUFF



Type: Numeric

Minimum value: 0

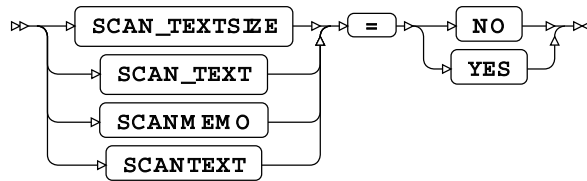
READBUFF



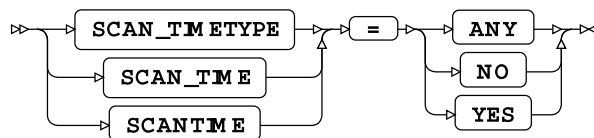
Type: Numeric

Minimum value: 0

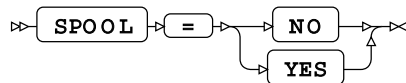
SCAN_TEXTSIZE



SCAN_TIMETYPE

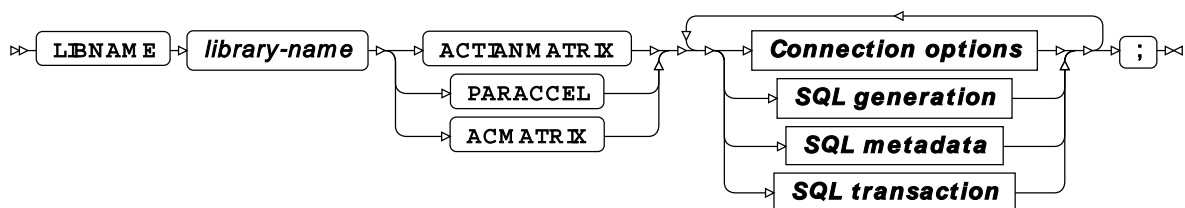


SPOOL



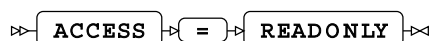
WPS Engine for Actian Matrix

ACTIANMATRIX

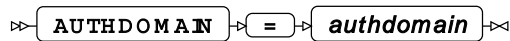


Connection options

ACCESS

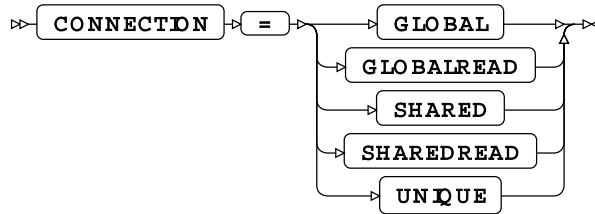


AUTHDOMAIN

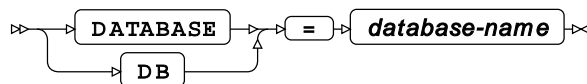


Type: String

CONNECTION

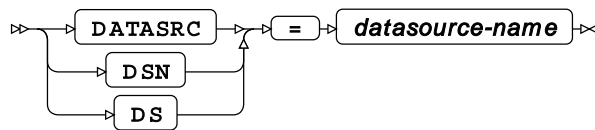


DATABASE



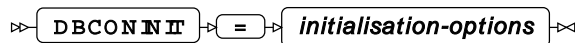
Type: String

DATASRC



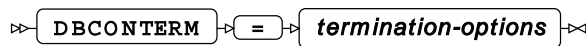
Type: String

DBCONINIT



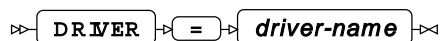
Type: String

DBCONTERM



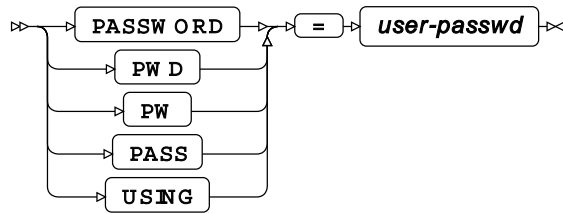
Type: String

DRIVER



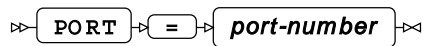
Type: String

PASSWORD



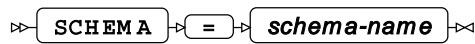
Type: String

PORT



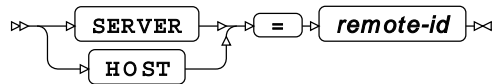
Type: String

SCHEMA



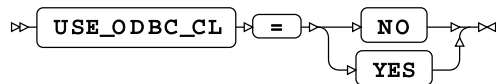
Type: String

SERVER

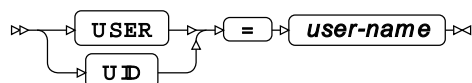


Type: String

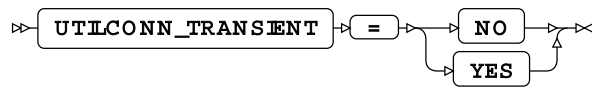
USE_ODBC_CL



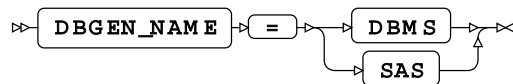
USER



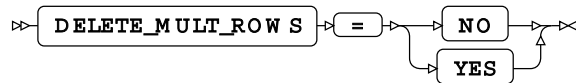
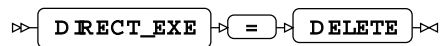
Type: String

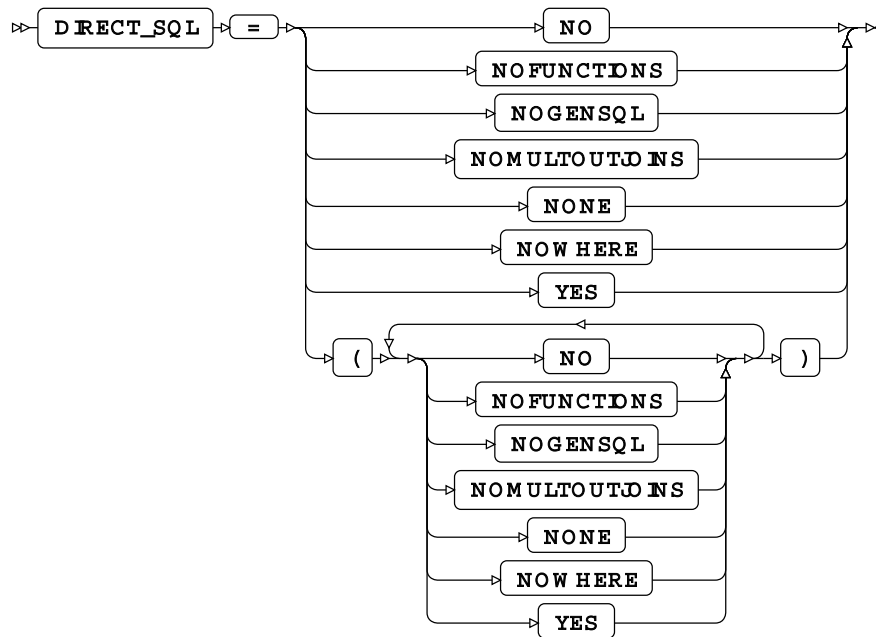
UTILCONN_TRANSIENT**SQL generation****DBCREATE_TABLE_OPTS**

Type: String

DBGEN_NAME

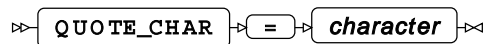
Default value: SAS

DELETE_MULT_ROWS**DIRECT_EXE****DIRECT_SQL**



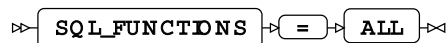
Default value: YES

QUOTE_CHAR

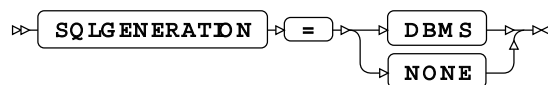


Type: String

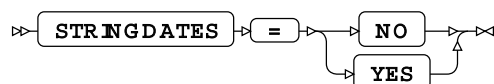
SQL_FUNCTIONS



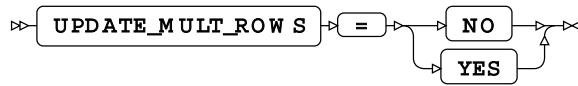
SQLGENERATION



STRINGDATES

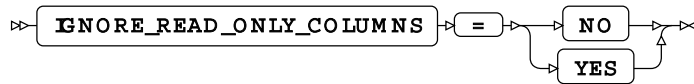


UPDATE_MULT_ROWS



SQL metadata

IGNORE_READ_ONLY_COLUMNS

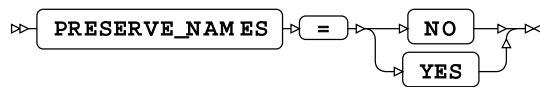


PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

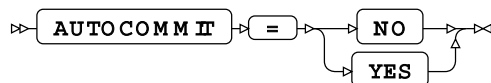
PRESERVE_TAB_NAMES



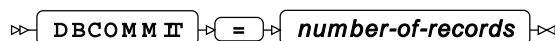
Default value: NO

SQL transaction

AUTOCOMMIT



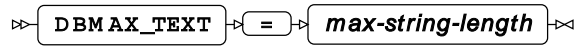
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

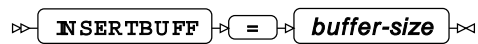


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF

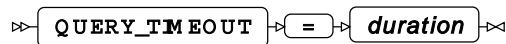


Type: Numeric

Minimum value: 1

Maximum value: 32767

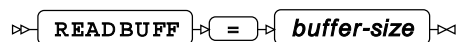
QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

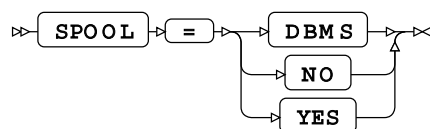


Type: Numeric

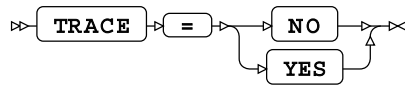
Minimum value: 1

Maximum value: 32767

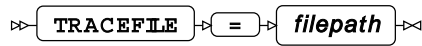
SPOOL



TRACE



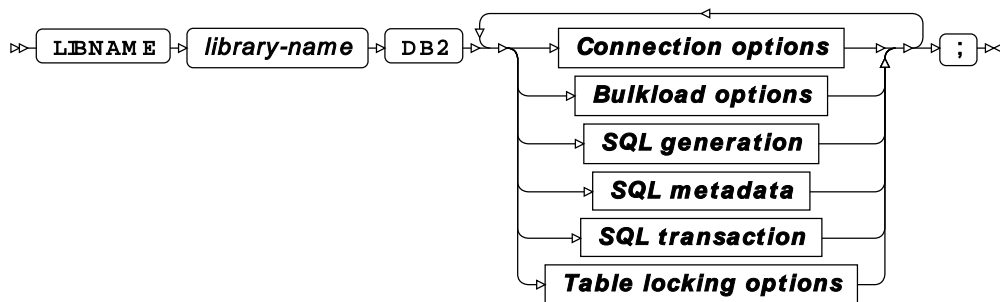
TRACEFILE



Type: String

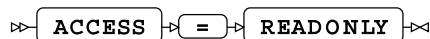
WPS Engine for DB2

DB2

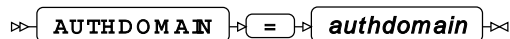


Connection options

ACCESS

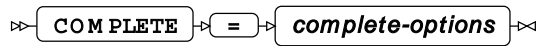


AUTHDOMAIN



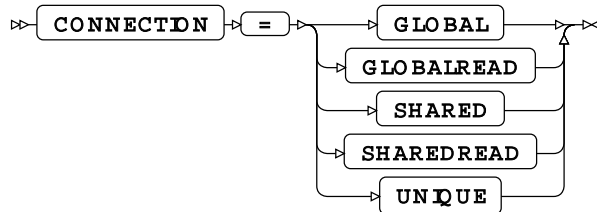
Type: String

COMPLETE

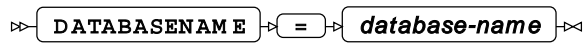


Type: String

CONNECTION

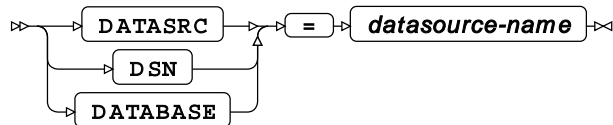


DATABASENAME



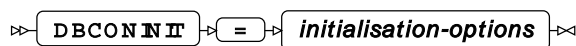
Type: String

DATASRC



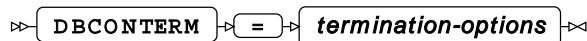
Type: String

DBCONINIT



Type: String

DBCONTERM

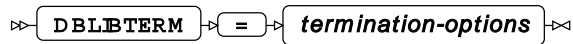


Type: String

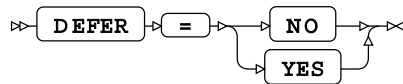
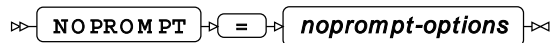
DBLIBINIT



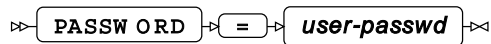
Type: String

DBLIBTERM

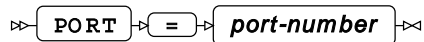
Type: String

DEFER**NOPROMPT**

Type: String

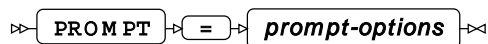
PASSWORD

Type: String

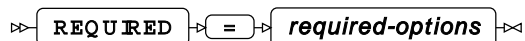
PORT

Minimum value: 1

Maximum value: 65535

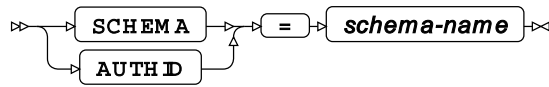
PROMPT

Type: String

REQUIRED

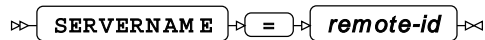
Type: String

SCHEMA



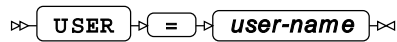
Type: String

SERVERNAME



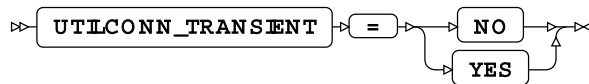
Type: String

USER



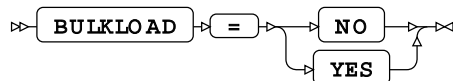
Type: String

UTILCONN_TRANSIENT

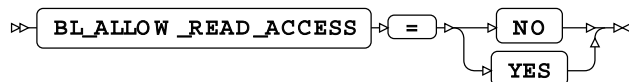


Bulkload options

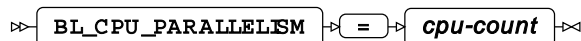
BULKLOAD



BL_ALLOW_READ_ACCESS

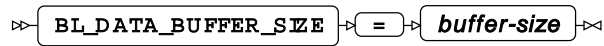


BL_CPU_PARALLELISM



Type: Numeric

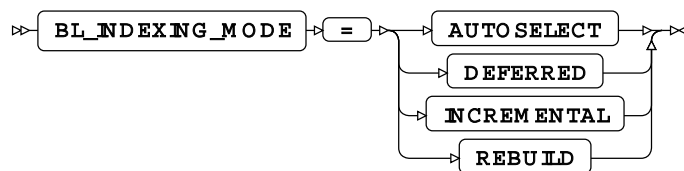
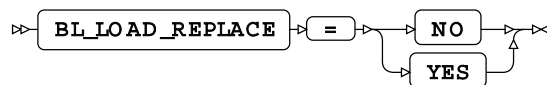
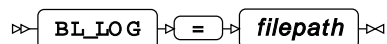
Maximum value: 30

BL_DATA_BUFFER_SIZE

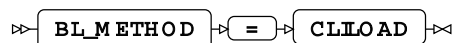
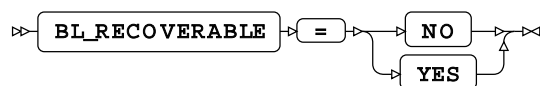
Type: Numeric

BL_DISK_PARALLELISM

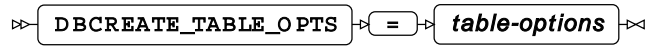
Type: Numeric

BL_INDEXING_MODE**BL_LOAD_REPLACE****BL_LOG**

Type: String

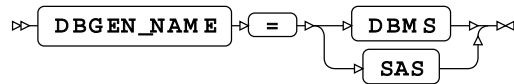
BL_METHOD**BL_RECOVERABLE****SQL generation**

DBCREATE_TABLE_OPTS



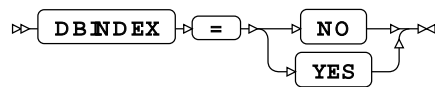
Type: String

DBGEN_NAME

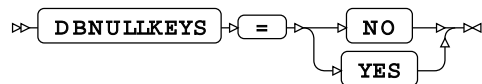


Default value: SAS

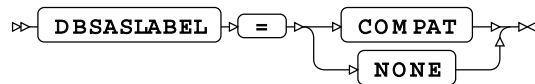
DBINDEX



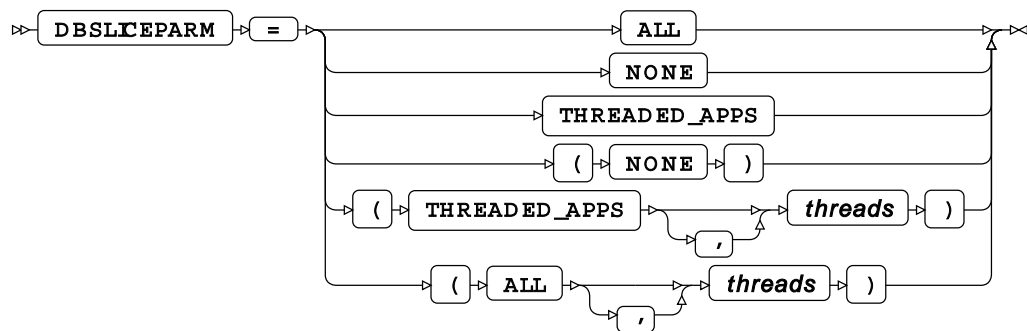
DBNULLKEYS



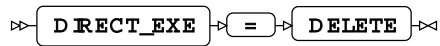
DBSASLABEL



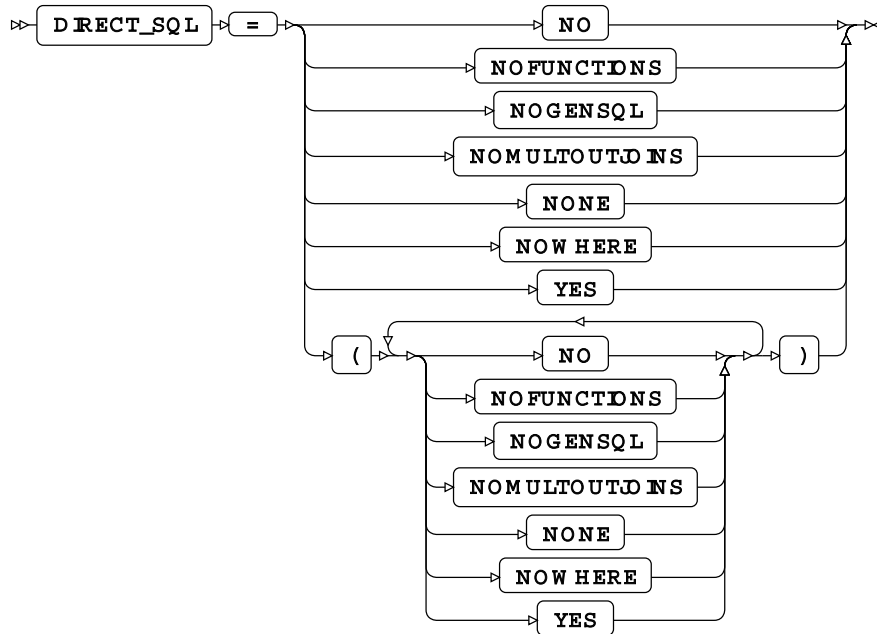
DBSLICEPARM



DIRECT_EXE

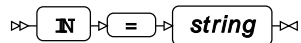


DIRECT_SQL



Default value: YES

IN

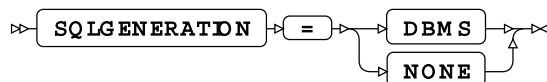


Type: String

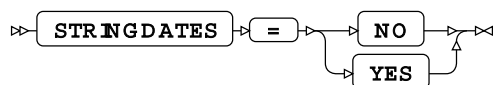
SQL_FUNCTIONS



SQLGENERATION

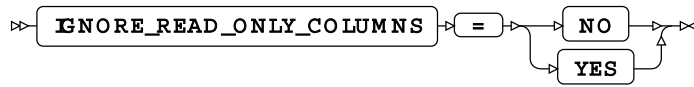


STRINGDATES



SQL metadata

IGNORE_READ_ONLY_COLUMNS

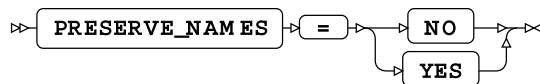


PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

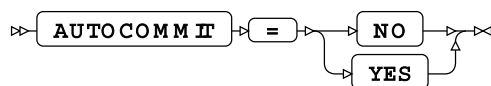
PRESERVE_TAB_NAMES



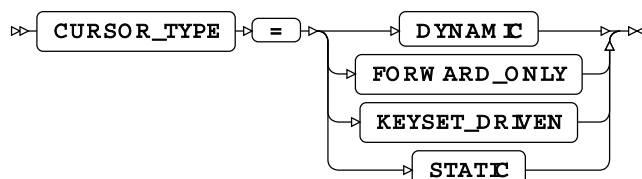
Default value: NO

SQL transaction

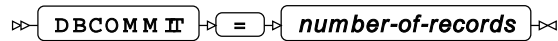
AUTOCOMMIT



CURSOR_TYPE



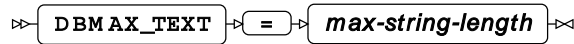
DBCMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

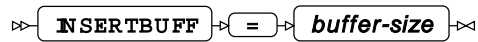


Type: Numeric

Minimum value: 1

Maximum value: 32767

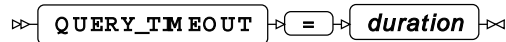
INSERTBUFF



Type: Numeric

Minimum value: 1

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF



Type: Numeric

Minimum value: 1

SPOOL

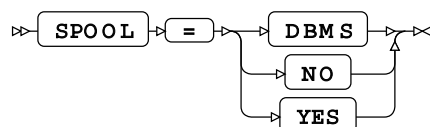
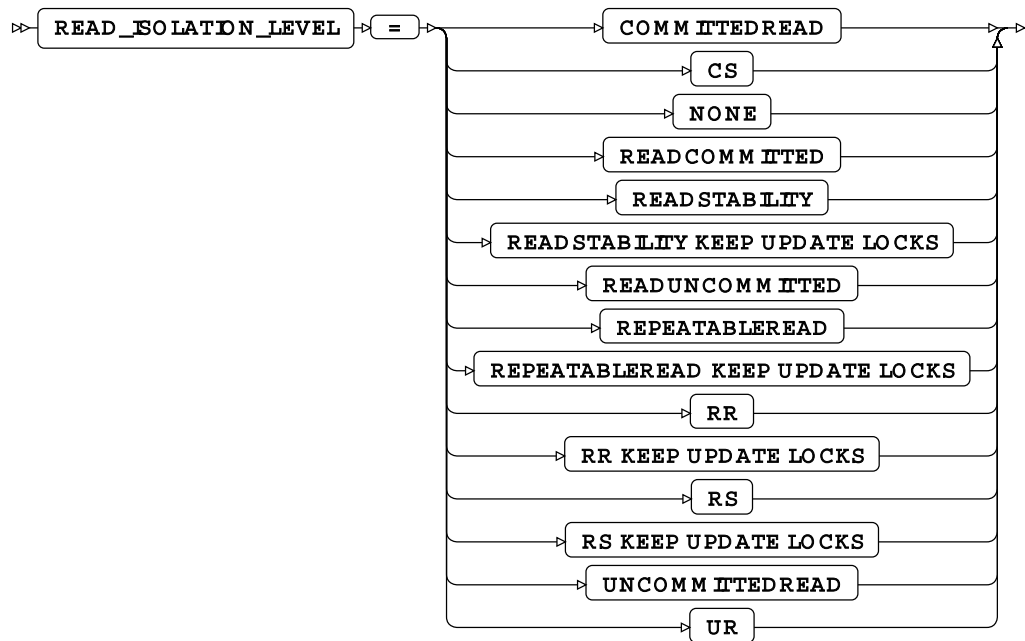
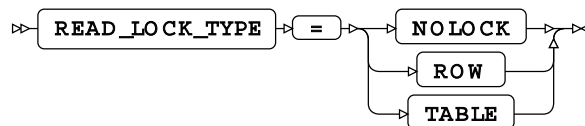


Table locking options

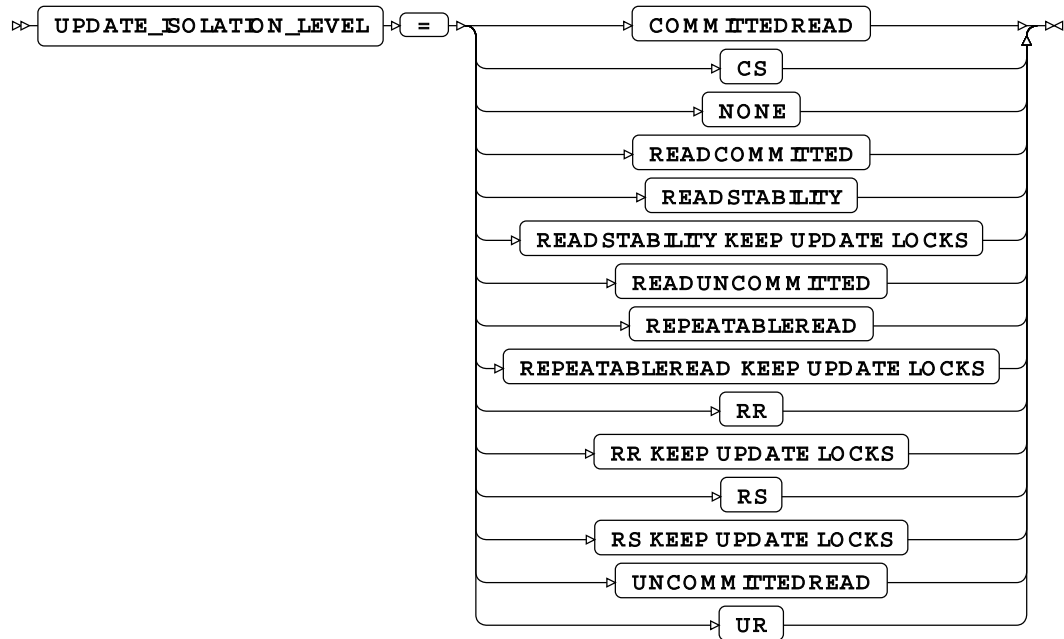
READ_ISOLATION_LEVEL



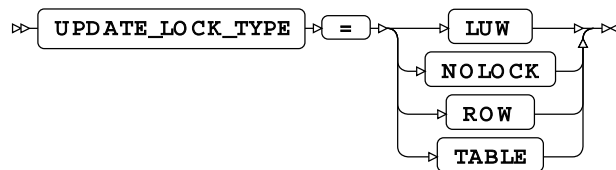
READ_LOCK_TYPE



UPDATE_ISOLATION_LEVEL

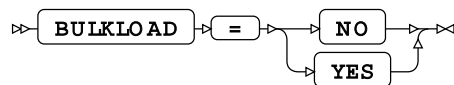


UPDATE_LOCK_TYPE

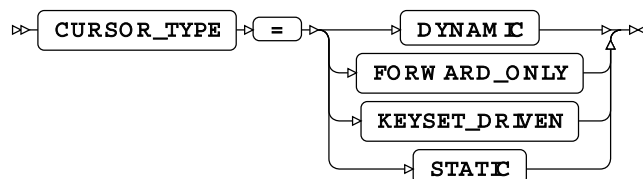


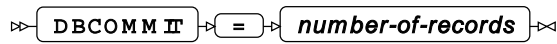
DB2 Dataset Options

BULKLOAD



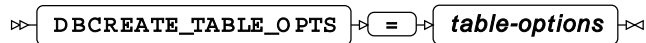
CURSOR_TYPE



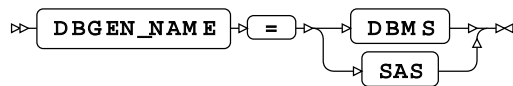
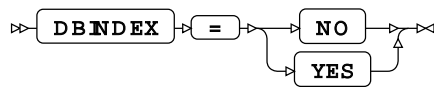
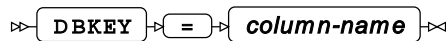
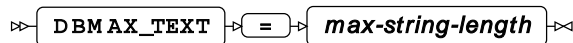
DBCOMMIT

Type: Numeric

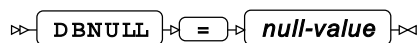
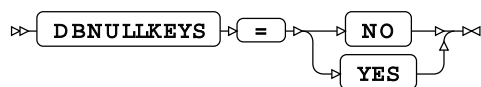
Minimum value: 0

DBCREATE_TABLE_OPTS

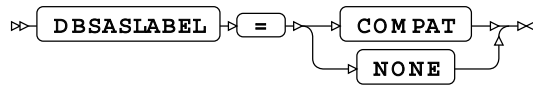
Type: String

DBGEN_NAME**DBINDEX****DBKEY****DBMAX_TEXT**

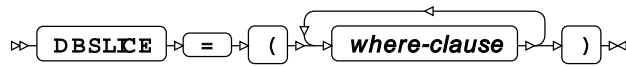
Type: Numeric

DBNULL**DBNULLKEYS**

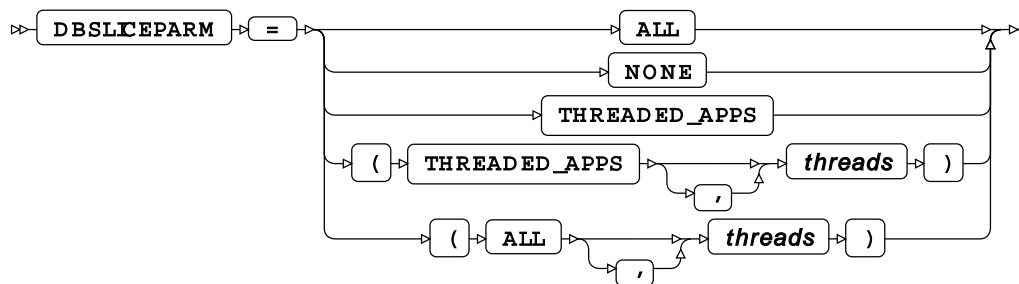
DBSASLABEL



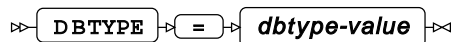
DBSLICE



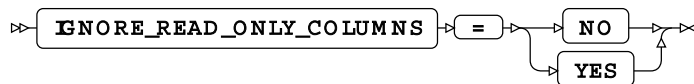
DBSLICEPARM



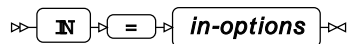
DBTYPE



IGNORE_READ_ONLY_COLUMNS

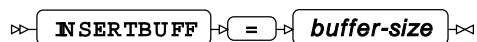


IN



Type: String

INSERTBUFF



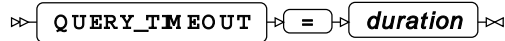
Minimum value: 1

PRESERVE_COL_NAMES



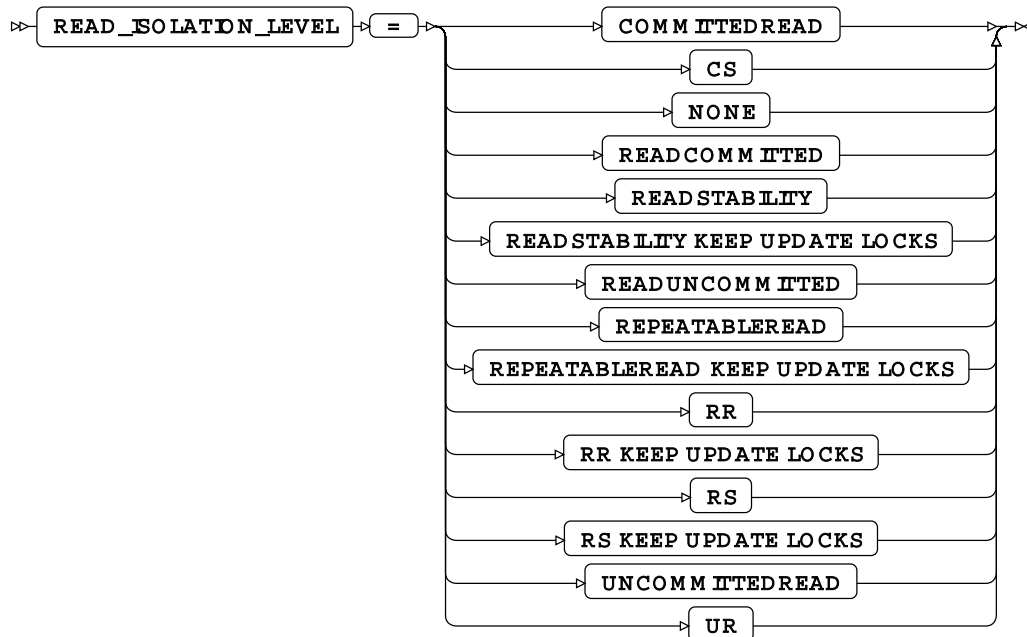
Default value: FALSE

QUERY_TIMEOUT

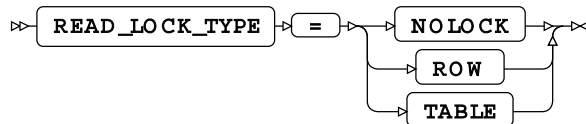


Minimum value: 0

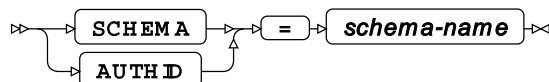
READ_ISOLATION_LEVEL



READ_LOCK_TYPE

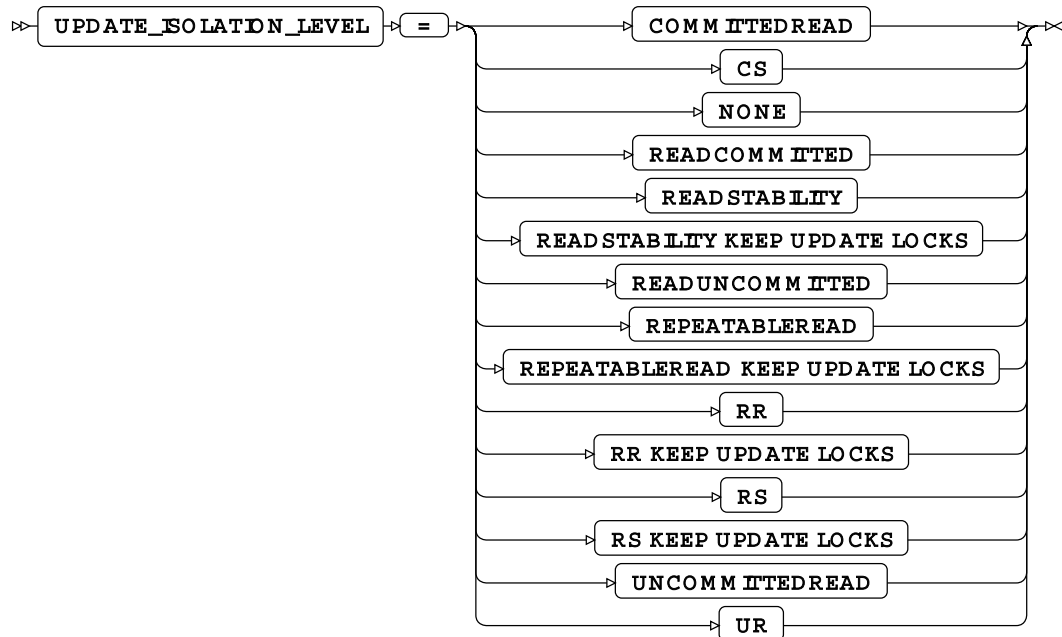


SCHEMA

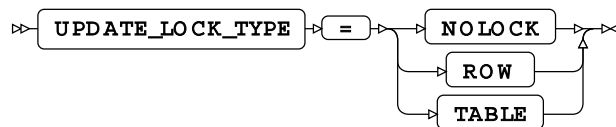


Type: String

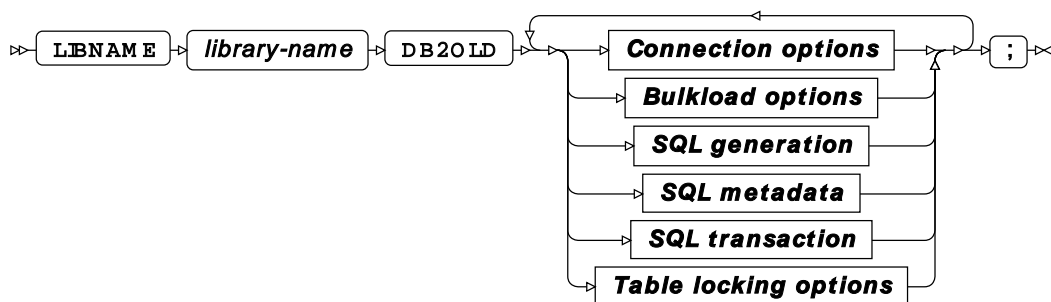
UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE

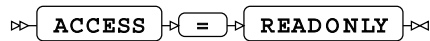


DB2OLD

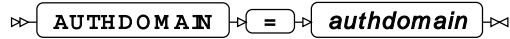


Connection options

ACCESS

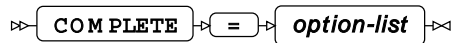


AUTHDOMAIN



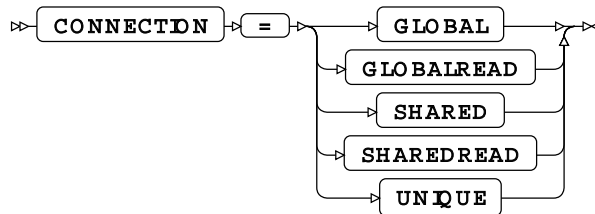
Type: String

COMPLETE

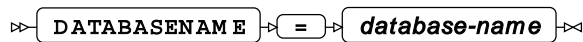


Type: String

CONNECTION

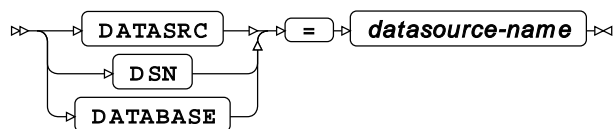


DATABASENAME



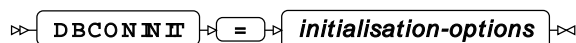
Type: String

DATASRC



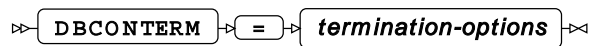
Type: String

DBCONINIT



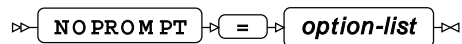
Type: String

DBCONTERM



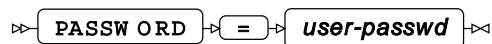
Type: String

NOPROMPT



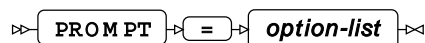
Type: String

PASSWORD



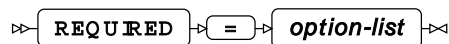
Type: String

PROMPT



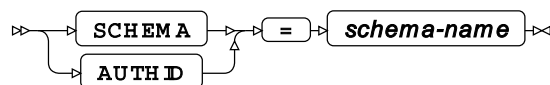
Type: String

REQUIRED



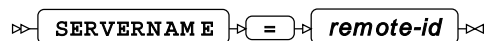
Type: String

SCHEMA



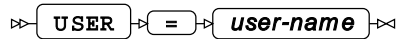
Type: String

SERVERNAME



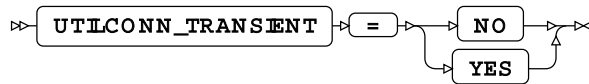
Type: String

USER



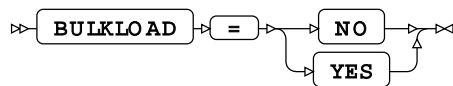
Type: String

UTILCONN_TRANSIENT

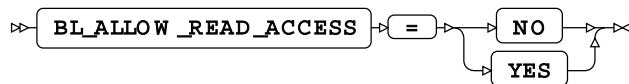


Bulkload options

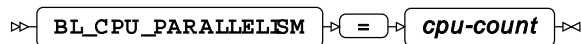
BULKLOAD



BL_ALLOW_READ_ACCESS

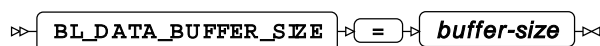


BL_CPU_PARALLELISM



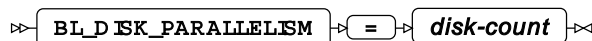
Type: Numeric

BL_DATA_BUFFER_SIZE



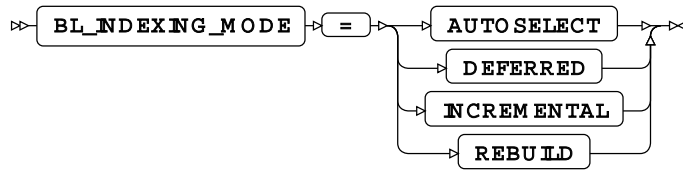
Type: Numeric

BL_DISK_PARALLELISM

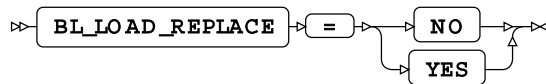


Type: Numeric

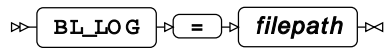
BL_INDEXING_MODE



BL_LOAD_REPLACE

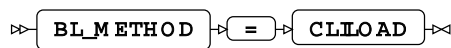


BL_LOG

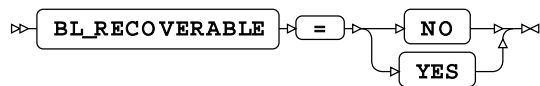


Type: String

BL_METHOD

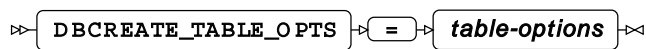


BL_RECOVERABLE



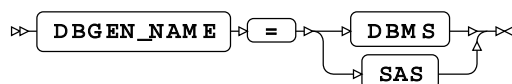
SQL generation

DBCREATE_TABLE_OPTS



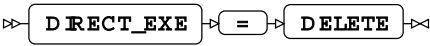
Type: String

DBGEN_NAME

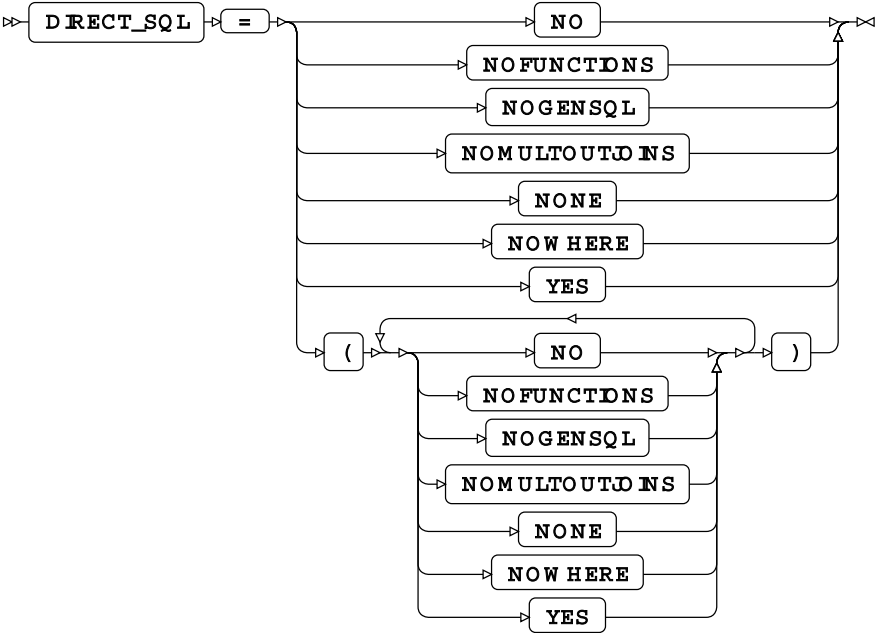


Default value: SAS

DIRECT_EXE

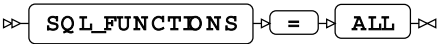


DIRECT_SQL

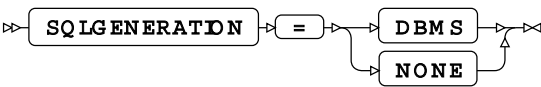


Default value: YES

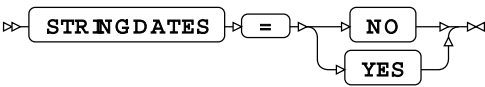
SQL_FUNCTIONS



SQLGENERATION



STRINGDATES



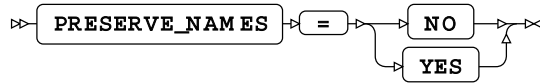
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

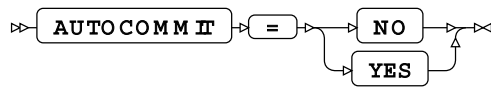
PRESERVE_TAB_NAMES



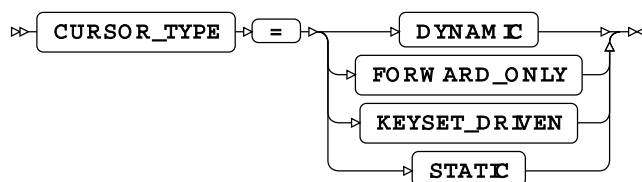
Default value: NO

SQL transaction

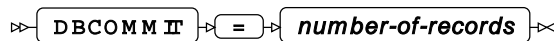
AUTOCOMMIT



CURSOR_TYPE



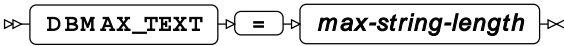
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

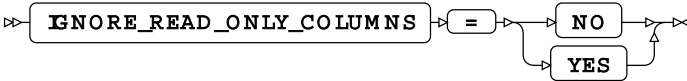


Type: Numeric

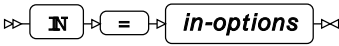
Minimum value: 1

Maximum value: 32767

IGNORE_READ_ONLY_COLUMNS

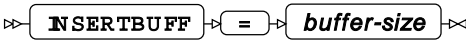


IN



Type: String

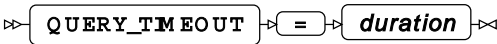
INSERTBUFF



Type: Numeric

Minimum value: 1

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF



Type: Numeric

Minimum value: 1

SPOOL

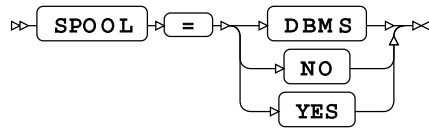
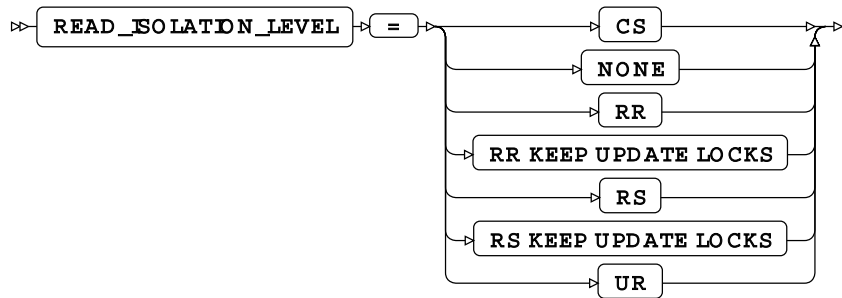
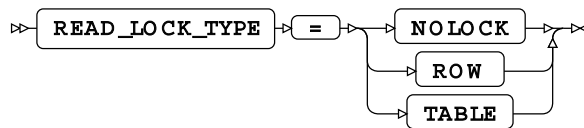


Table locking options

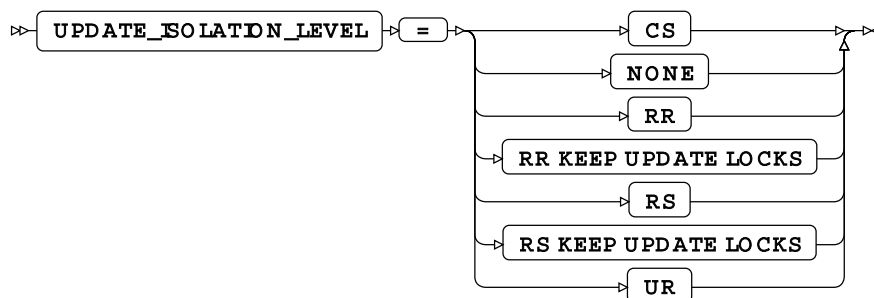
READ_ISOLATION_LEVEL



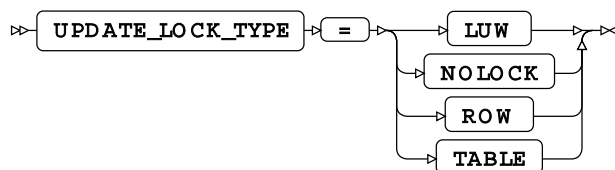
READ_LOCK_TYPE



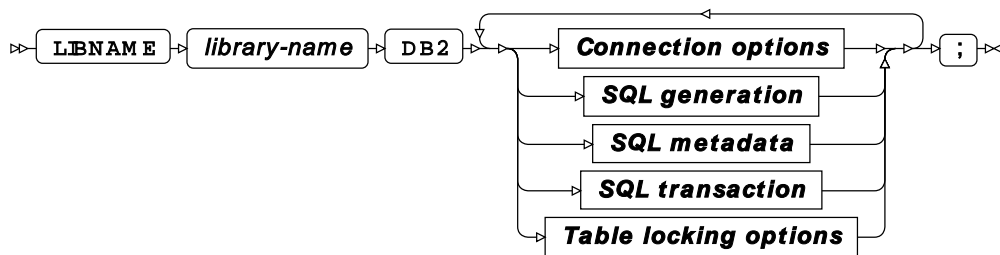
UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE

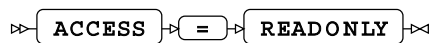


DB2 (for z/OS)



Connection options

ACCESS

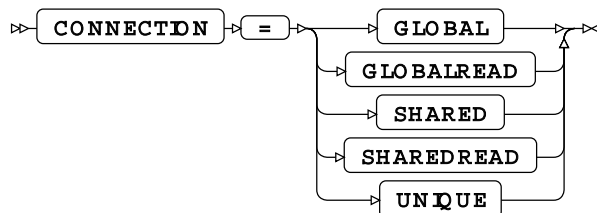


AUTHDOMAIN



Type: String

CONNECTION

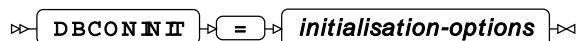


DATABASENAME



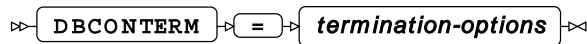
Type: String

DBCONINIT



Type: String

DBCONTERM



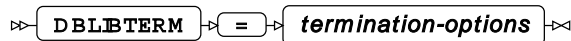
Type: String

DBLIBINIT



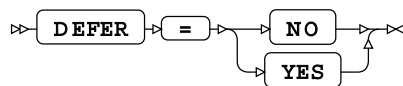
Type: String

DBLIBTERM

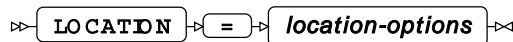


Type: String

DEFER

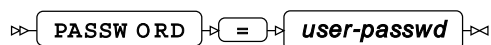


LOCATION



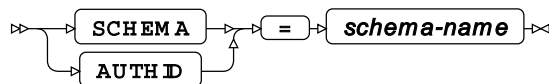
Type: String

PASSWORD



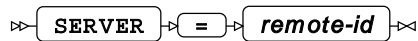
Type: String

SCHEMA



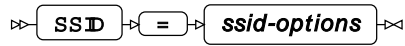
Type: String

SERVER



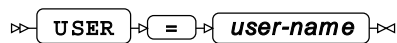
Type: String

SSID



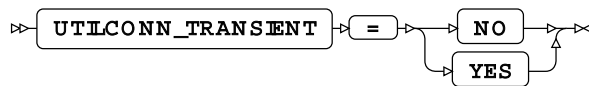
Type: String

USER



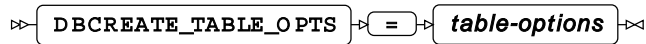
Type: String

UTILCONN_TRANSIENT



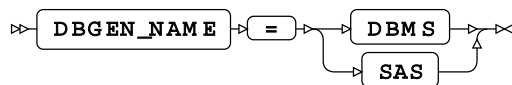
SQL generation

DBCREATE_TABLE_OPTS



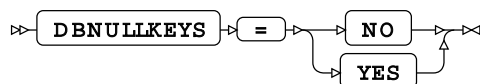
Type: String

DBGEN_NAME

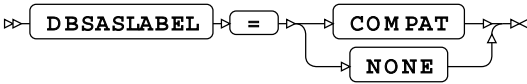


Default value: SAS

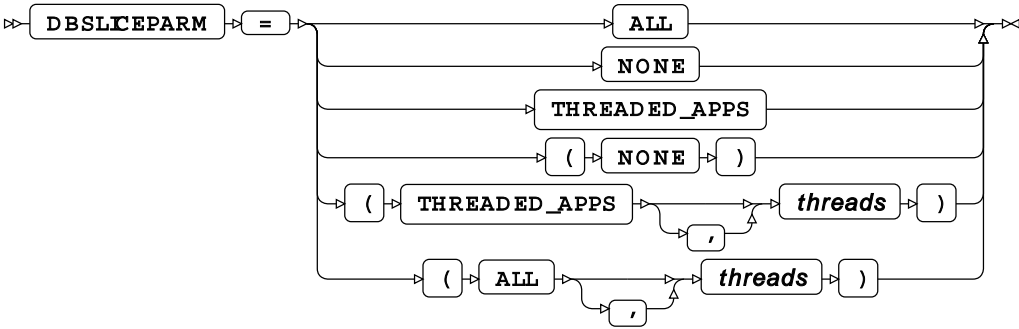
DBNULLKEYS



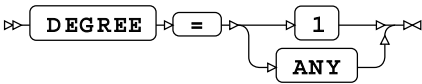
DBSASLABEL



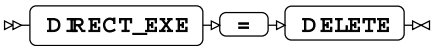
DBSLICEPARM



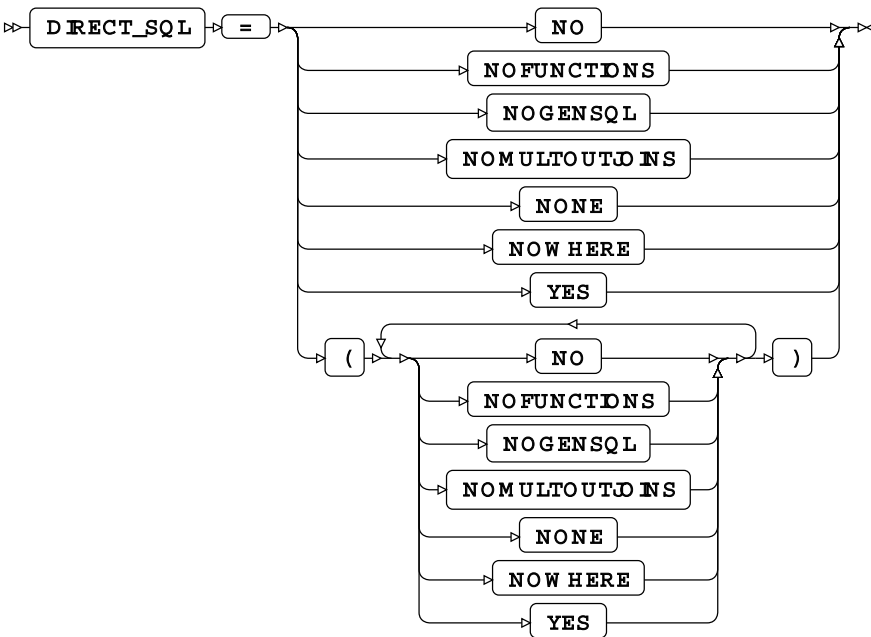
DEGREE



DIRECT_EXE

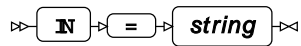


DIRECT_SQL



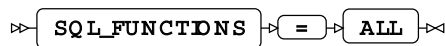
Default value: YES

IN

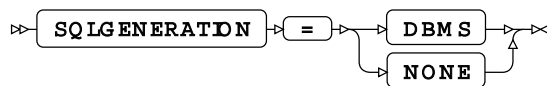


Type: String

SQL_FUNCTIONS



SQLGENERATION



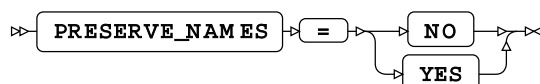
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



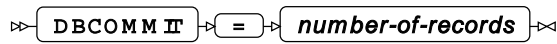
Default value: NO

PRESERVE_TAB_NAMES



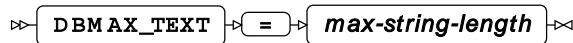
Default value: NO

SQL transaction

DBCOMMIT

Type: Numeric

Minimum value: 0

DBMAX_TEXT

Type: Numeric

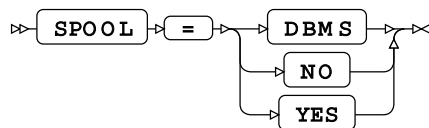
Minimum value: 1

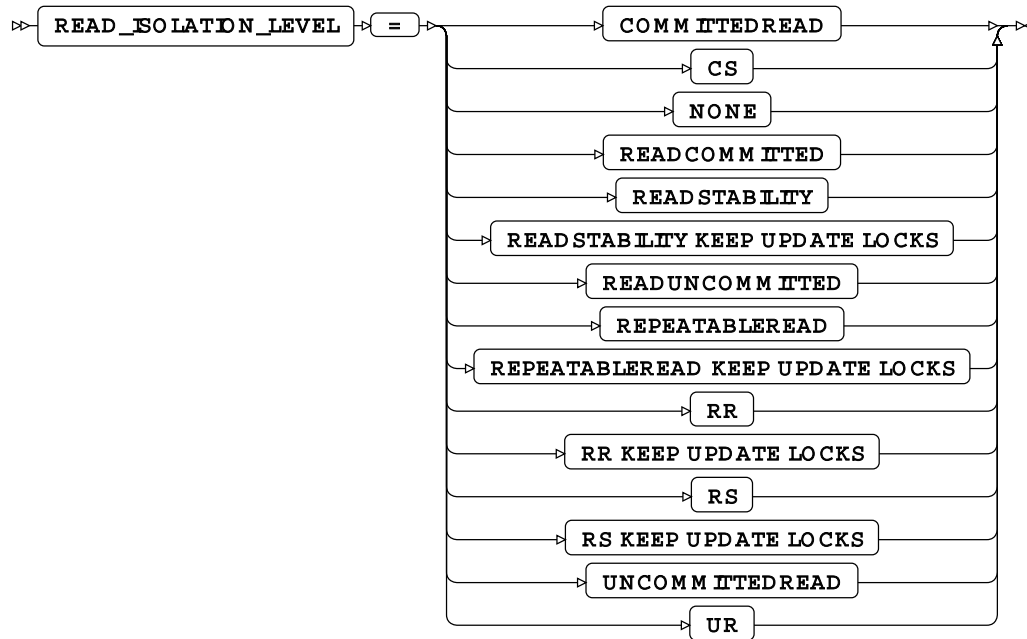
Maximum value: 32767

READBUFF

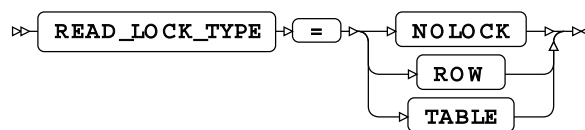
Type: Numeric

Minimum value: 1

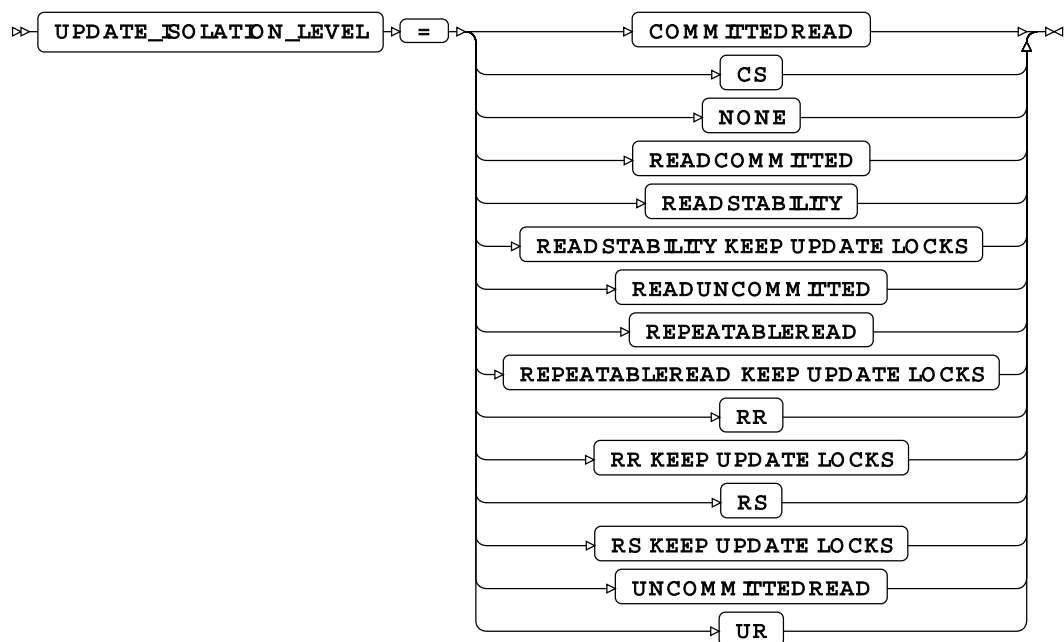
SPOOL**Table locking options****READ_ISOLATION_LEVEL**

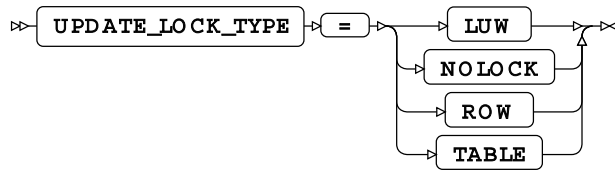


READ_LOCK_TYPE

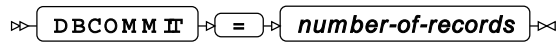


UPDATE_ISOLATION_LEVEL



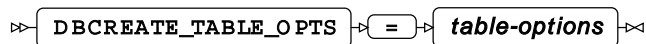
UPDATE_LOCK_TYPE

DB2 Dataset Options

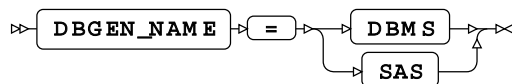
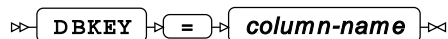
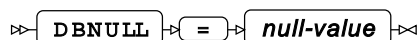
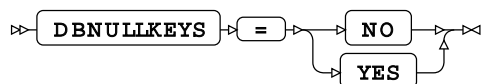
DBCMMIT

Type: Numeric

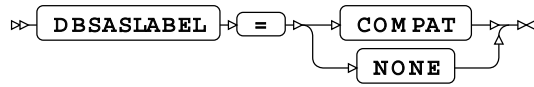
Minimum value: 0

DBCREATE_TABLE_OPTS

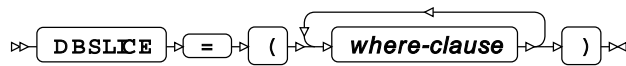
Type: String

DBGEN_NAME**DBKEY****DBNULL****DBNULLKEYS**

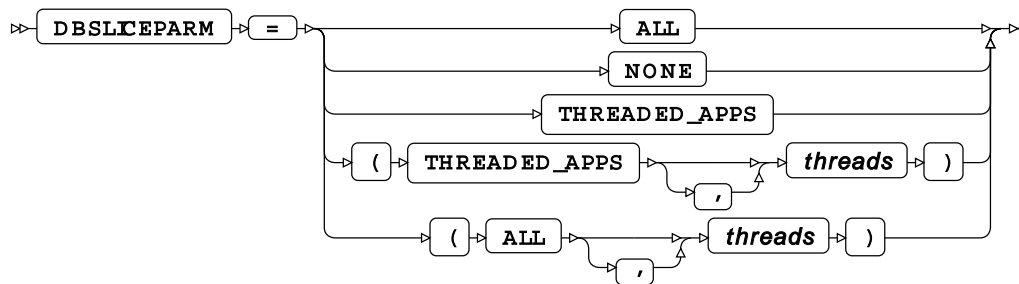
DBSASLABEL



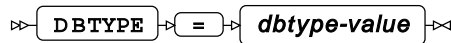
DBSLICE



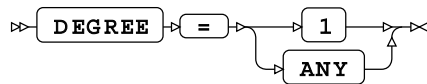
DBSLICEPARM



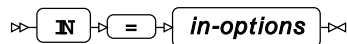
DBTYPE



DEGREE

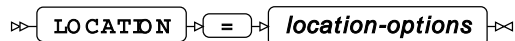


IN



Type: String

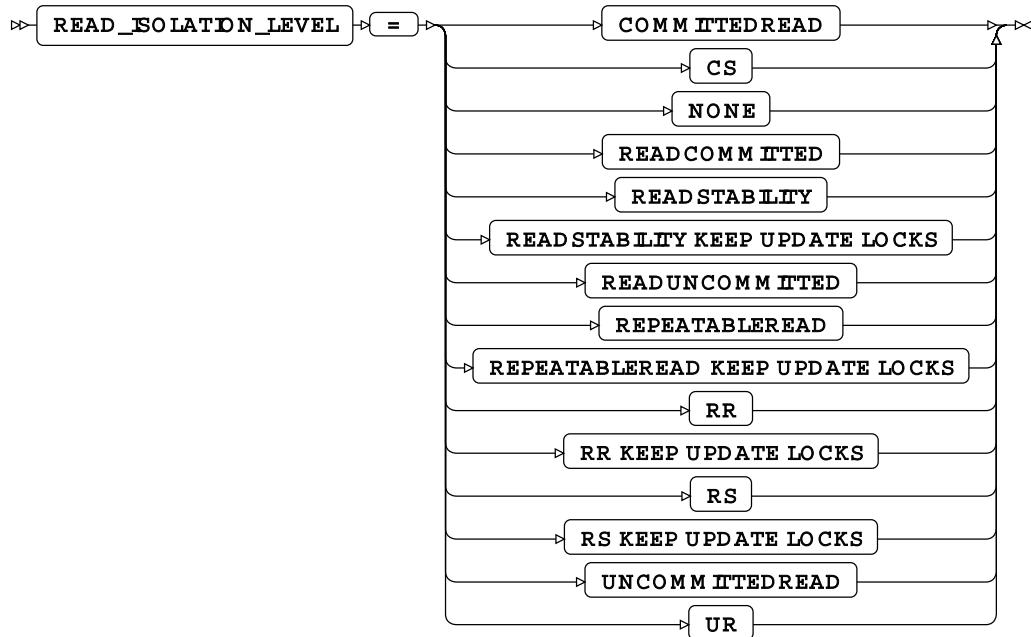
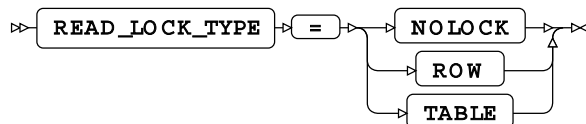
LOCATION



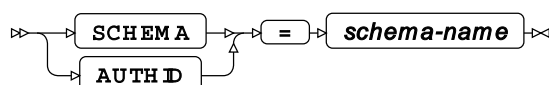
Type: String

PRESERVE_COL_NAMES

Default value: FALSE

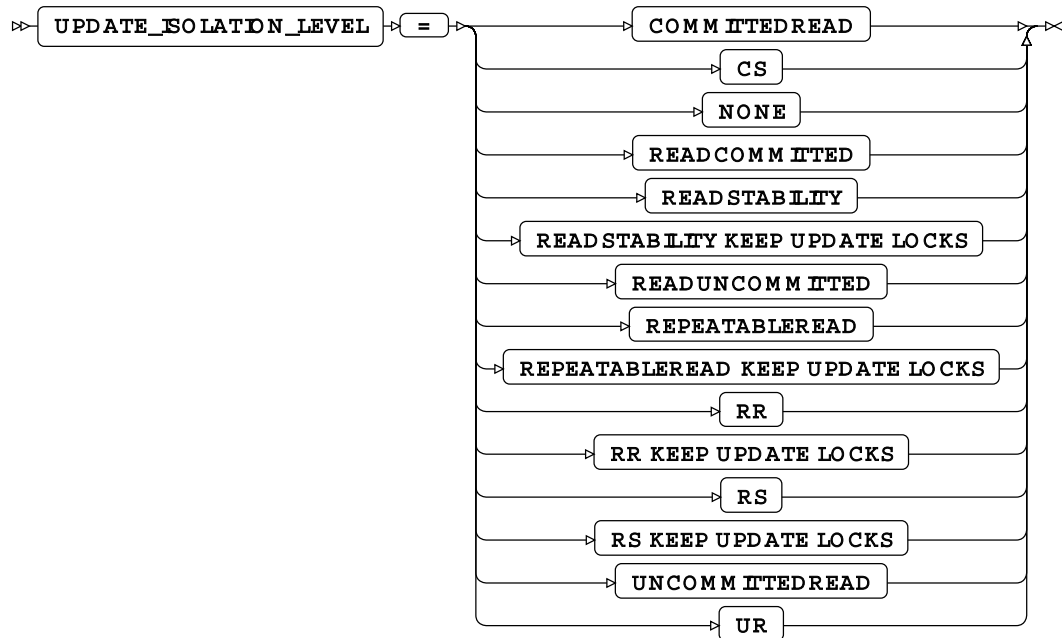
READ_ISOLATION_LEVEL**READ_LOCK_TYPE****READBUFF**

Minimum value: 1

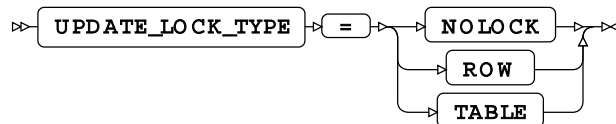
SCHEMA

Type: String

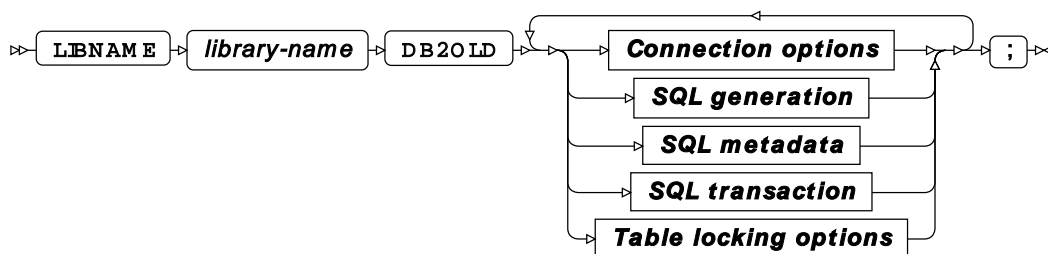
UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE

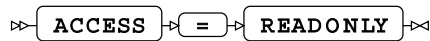


DB2OLD (for z/OS)

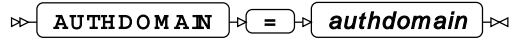


Connection options

ACCESS

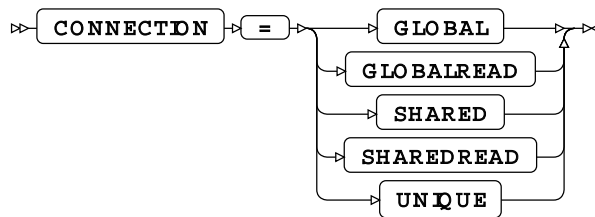


AUTHDOMAIN

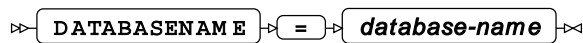


Type: String

CONNECTION

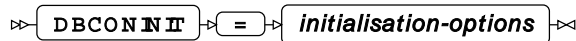


DATABASENAME



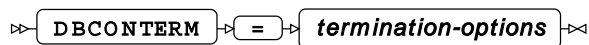
Type: String

DBCONINIT



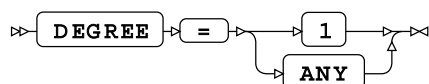
Type: String

DBCONTERM

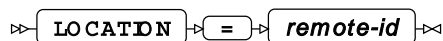


Type: String

DEGREE

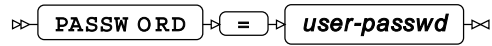


LOCATION



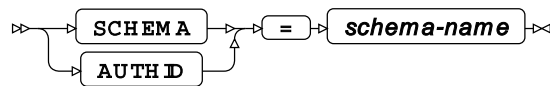
Type: String

PASSWORD



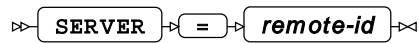
Type: String

SCHEMA



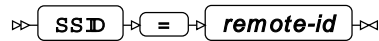
Type: String

SERVER



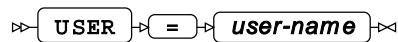
Type: String

SSID



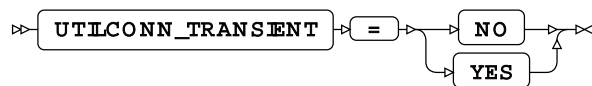
Type: String

USER



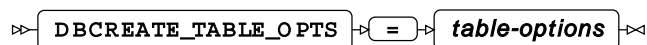
Type: String

UTILCONN_TRANSIENT



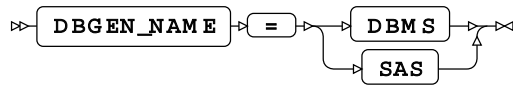
SQL generation

DBCREATE_TABLE_OPTS



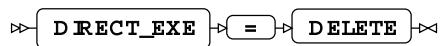
Type: String

DBGEN_NAME

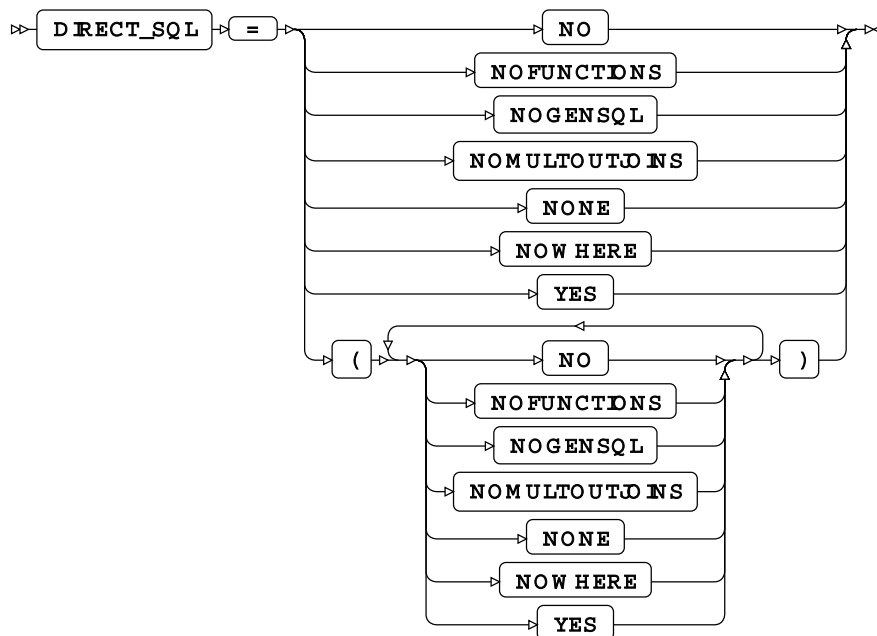


Default value: SAS

DIRECT_EXE

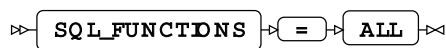


DIRECT_SQL

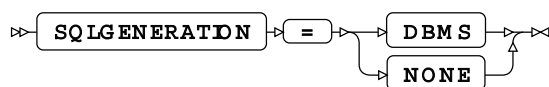


Default value: YES

SQL_FUNCTIONS



SQLGENERATION



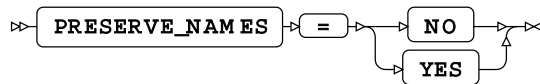
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

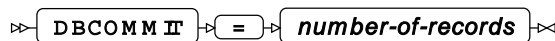
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

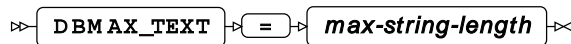
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

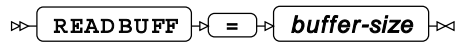


Type: Numeric

Minimum value: 1

Maximum value: 32767

READBUFF



Type: Numeric

Minimum value: 1

SPOOL

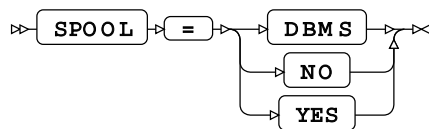
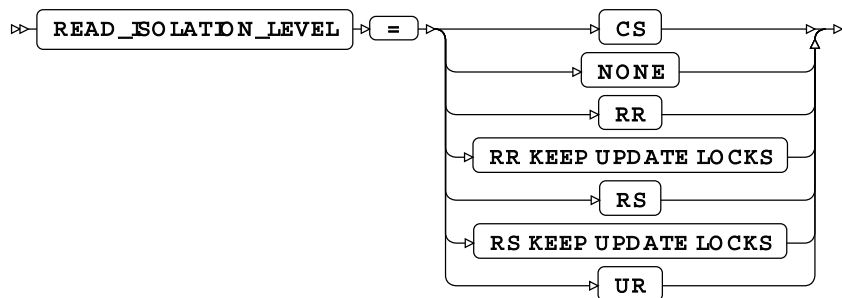
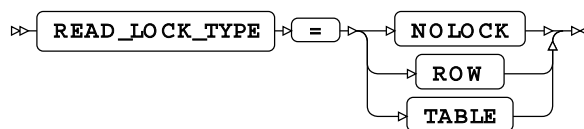


Table locking options

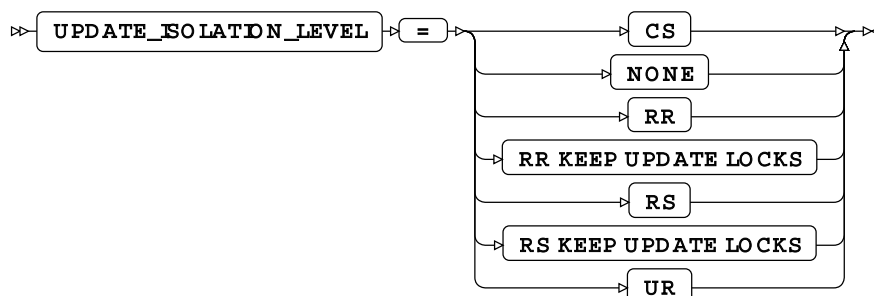
READ_ISOLATION_LEVEL



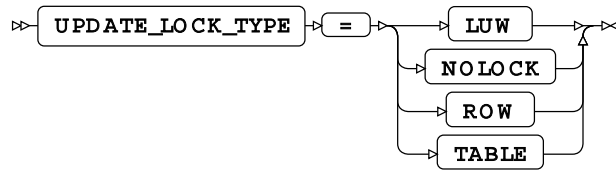
READ_LOCK_TYPE



UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE

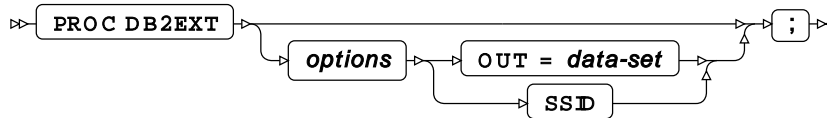


DB2EXT Procedure

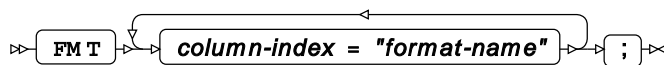
Supported statements

- `PROC DB2EXT` [\(page 3553\)](#)
- `FMT` [\(page 3553\)](#)
- `RENAME` [\(page 3553\)](#)
- `SELECT` [\(page 3554\)](#)
- `EXIT` [\(page 3554\)](#)

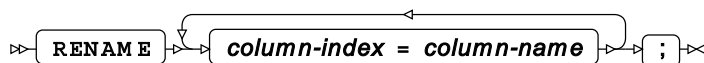
PROC DB2EXT



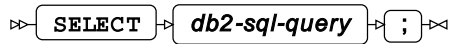
FMT



RENAME



SELECT

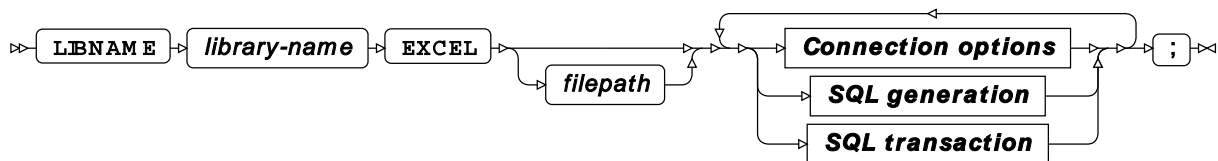


EXIT



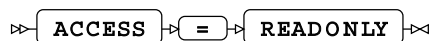
WPS Engine for Excel

EXCEL

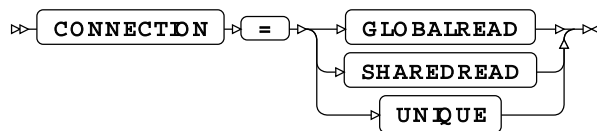


Connection options

ACCESS

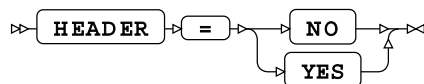


CONNECTION

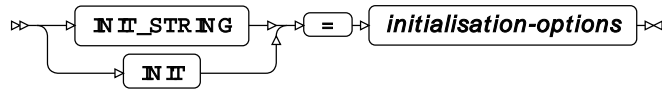


Default value: SHAREDREAD

HEADER

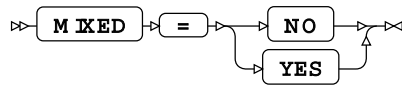


INIT_STRING

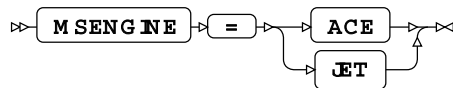


Type: String

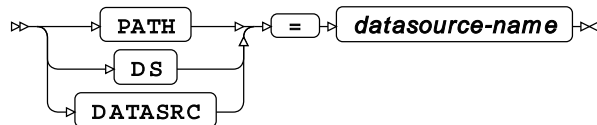
MIXED



MSENGINE

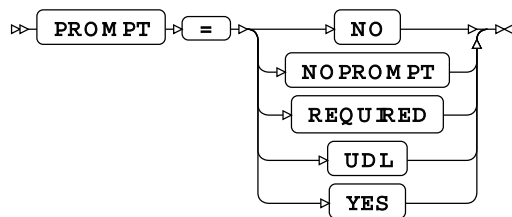


PATH

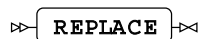


Type: String

PROMPT

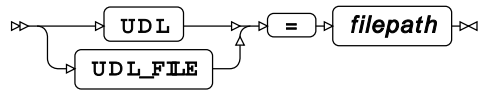


REPLACE



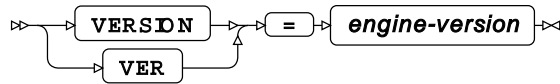
Type: Keyword

UDL



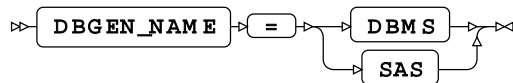
Type: String

VERSION

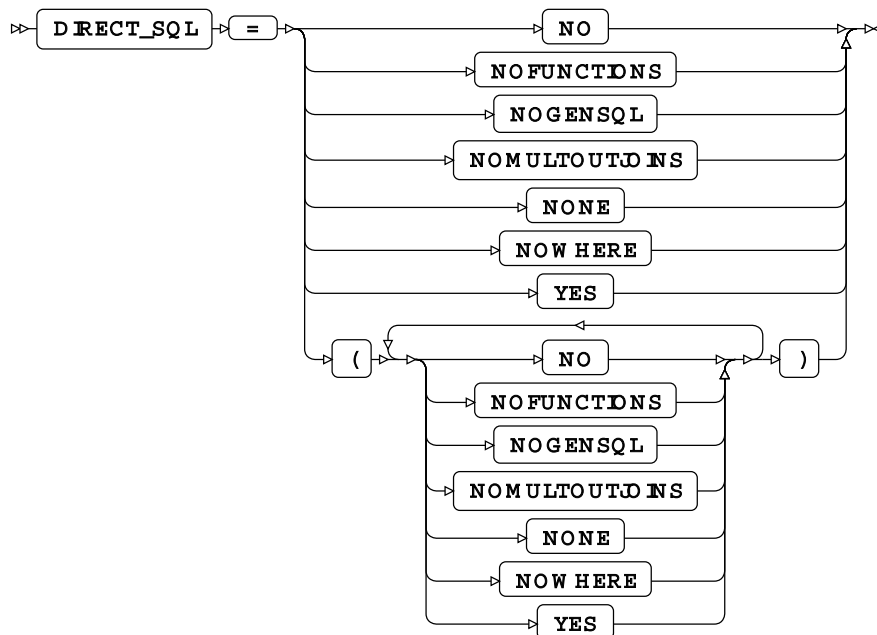


SQL generation

DBGEN_NAME



DIRECT_SQL

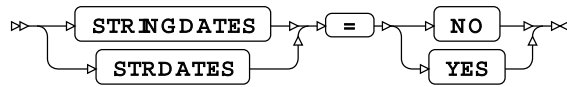


LABEL

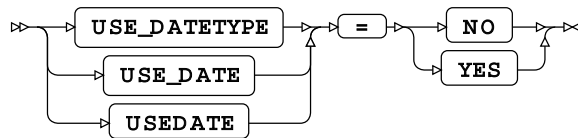


Type: Keyword

STRINGDATES



USE_DATETYPE



SQL transaction

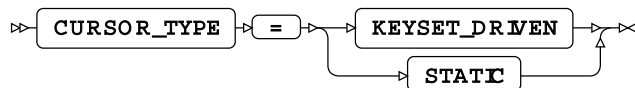
COMMAND_TIMEOUT



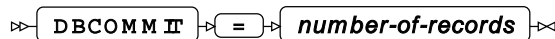
Type: Numeric

Minimum value: 0

CURSOR_TYPE



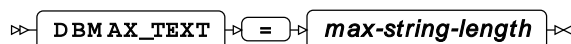
DBCMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

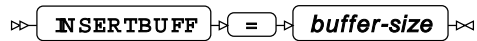


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF



Type: Numeric

Minimum value: 0

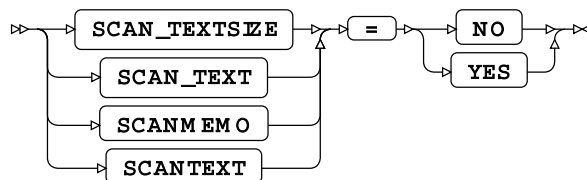
READBUFF



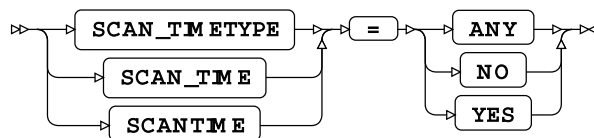
Type: Numeric

Minimum value: 0

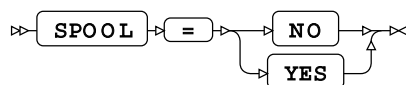
SCAN_TEXTSIZE



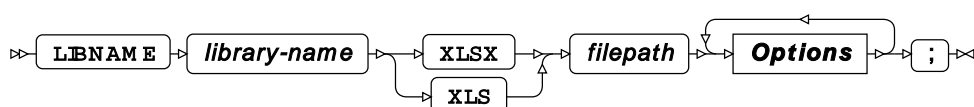
SCAN_TIMETYPE



SPOOL

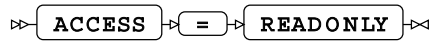


XLSX

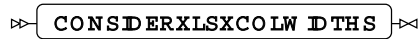


Options

ACCESS

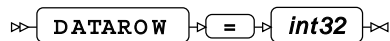


CONSIDERXLSXCOLWIDTHS



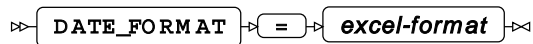
Type: Keyword

DATAROW



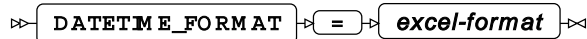
Type: Numeric

DATE_FORMAT



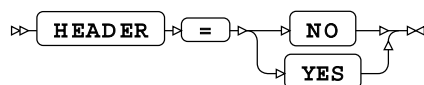
Type: String

DATETIME_FORMAT



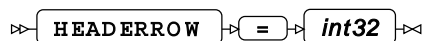
Type: String

HEADER



Default value: YES

HEADERROW



Type: Numeric

LABEL

» **LABEL** «

Type: Keyword

NOCONSIDERXLSXCOLWIDTHS

» **NOCONSIDERXLSXCOLWIDTHS** «

Type: Keyword

NOREPLACE

» **NOREPLACE** «

Type: Keyword

REPLACE

» **REPLACE** «

Type: Keyword

TIME_FORMAT

» **TIME_FORMAT** » = » *excel-format* «

Type: String

XLSX Dataset Options

CONSIDERXLSXCOLWIDTHS

» **CONSIDERXLSXCOLWIDTHS** «

Type: Keyword

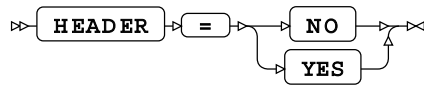
DATAROW

» **DATAROW** » = » *None* «

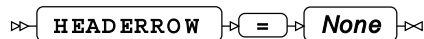
Type: Numeric

Minimum value: 1

HEADER



HEADERROW



Type: Numeric

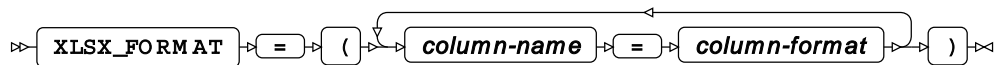
Minimum value: 1

NOCONSIDERXLSXCOLWIDTHS



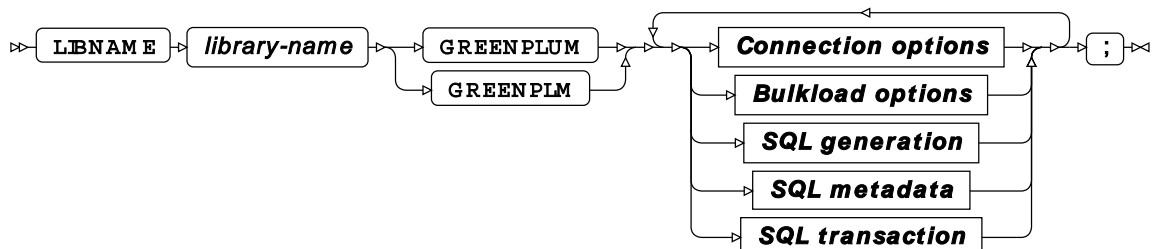
Type: Keyword

XLSX_FORMAT



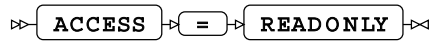
WPS Engine for Greenplum

GREENPLUM

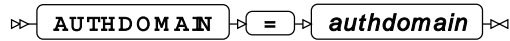


Connection options

ACCESS

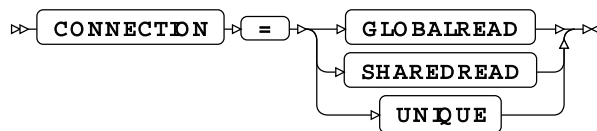


AUTHDOMAIN

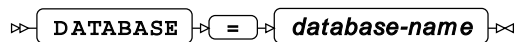


Type: String

CONNECTION

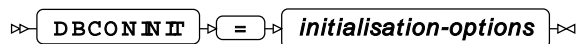


DATABASE



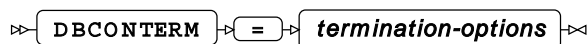
Type: String

DBCONINIT



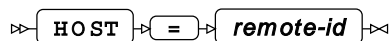
Type: String

DBCONTERM



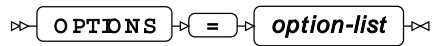
Type: String

HOST



Type: String

OPTIONS



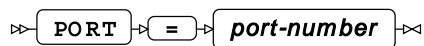
Type: String

PASSWORD



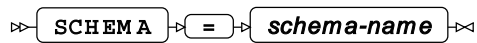
Type: String

PORT



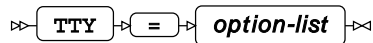
Type: String

SCHEMA



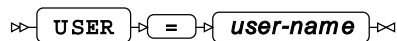
Type: String

TTY



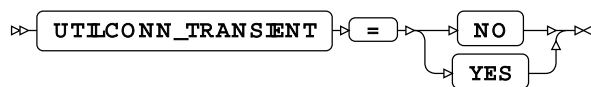
Type: String

USER



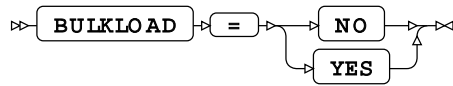
Type: String

UTILCONN_TRANSIENT

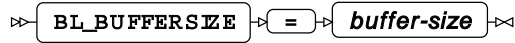


Bulkload options

BULKLOAD

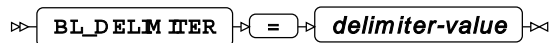


BL_BUFFERSIZE



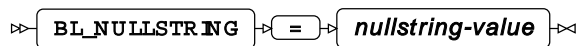
Type: String

BL_DELIMITER



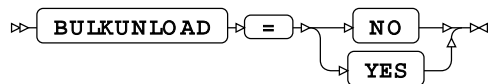
Type: String

BL_NULLSTRING



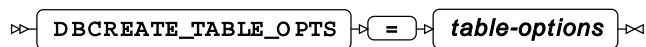
Type: String

BULKUNLOAD



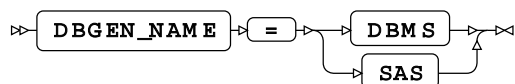
SQL generation

DBCREATE_TABLE_OPTS



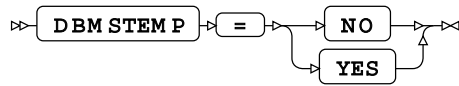
Type: String

DBGEN_NAME

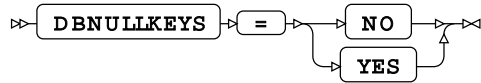


Default value: SAS

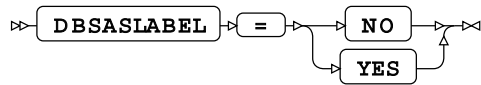
DBMSTEMP



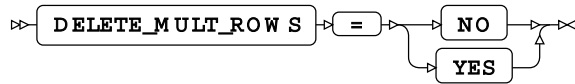
DBNULLKEYS



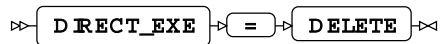
DBSASLABEL



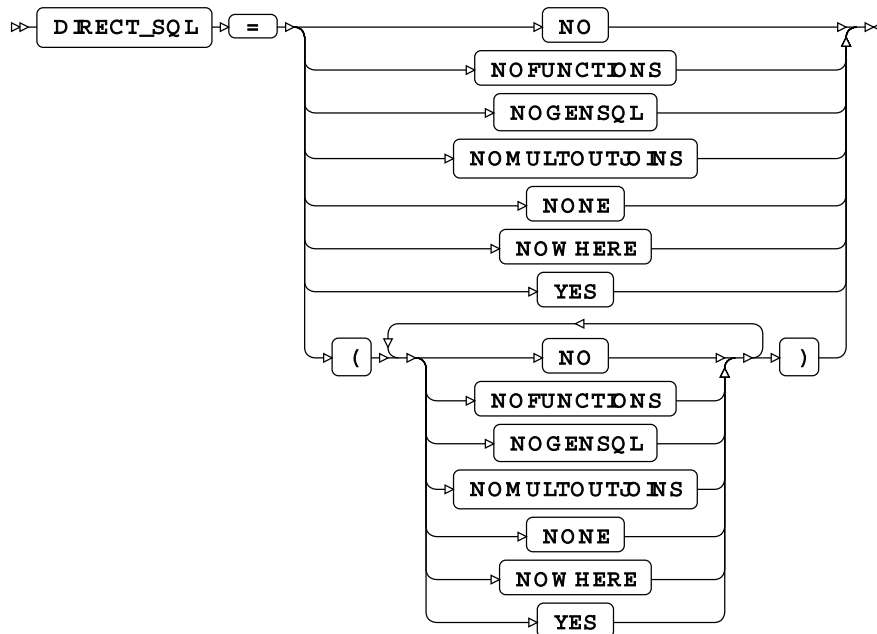
DELETE_MULT_ROWS



DIRECT_EXE



DIRECT_SQL



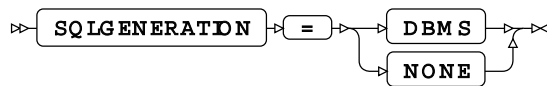
Default value: YES

SQL_FUNCTIONS

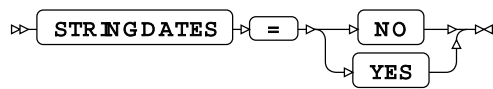


Default value: ALL

SQLGENERATION

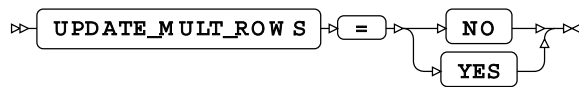


STRINGDATES



Default value: YES

UPDATE_MULT_ROWS



SQL metadata

IGNORE_READ_ONLY_COLUMNS



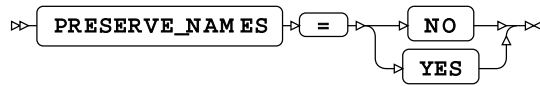
Default value: NO

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

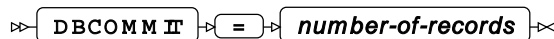
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

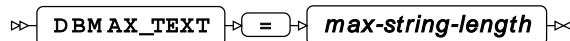
DBCMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

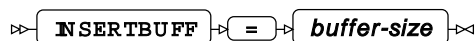


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF



Type: Numeric

Minimum value: 1

Maximum value: 32767

READBUFF



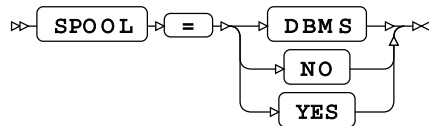
Type: Numeric

Minimum value: 1

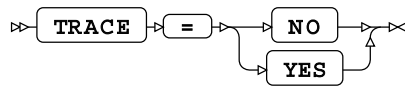
Maximum value: 32767

Default value: 1

SPOOL

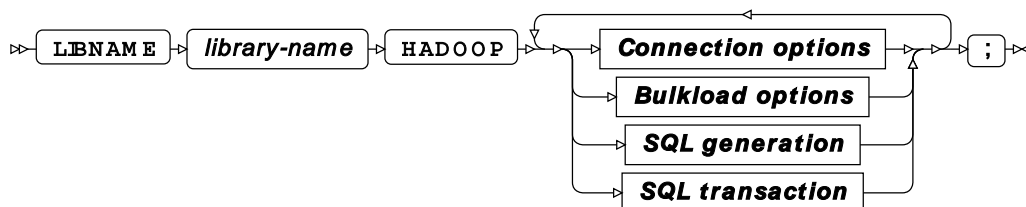


TRACE



WPS Engine for Hadoop

HADOOP

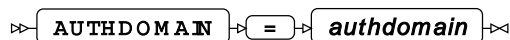


Connection options

ACCESS

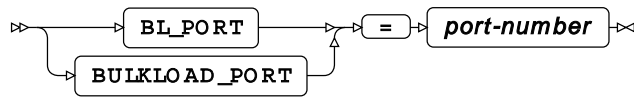


AUTHDOMAIN



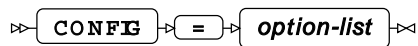
Type: String

BL_PORT



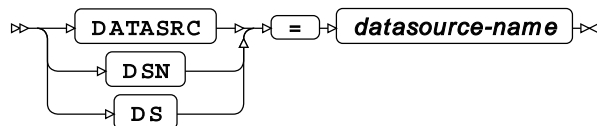
Type: Numeric

CONFIG



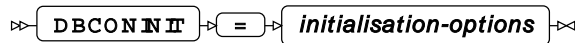
Type: String

DATASRC



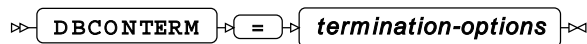
Type: String

DBCONINIT



Type: String

DBCONTERM



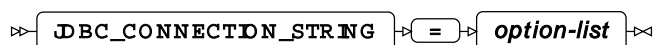
Type: String

HIVE_PRINCIPAL



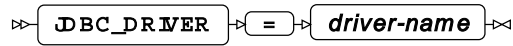
Type: String

JDBC_CONNECTION_STRING



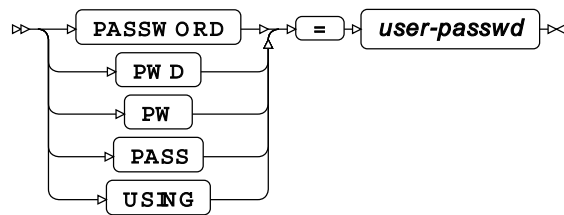
Type: String

JDBC_DRIVER



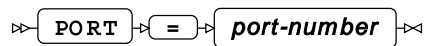
Type: String

PASSWORD



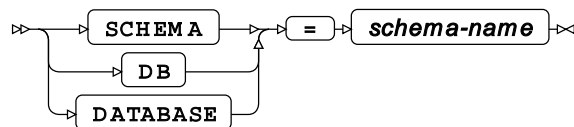
Type: String

PORT



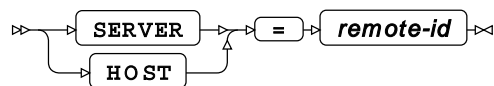
Type: Numeric

SCHEMA



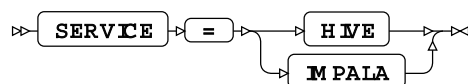
Type: String

SERVER

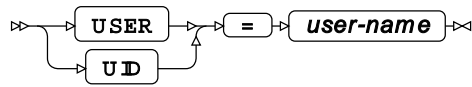


Type: String

SERVICE



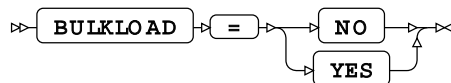
USER



Type: String

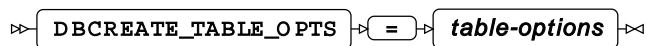
Bulkload options

BULKLOAD



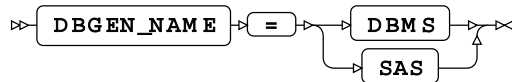
SQL generation

DBCREATE_TABLE_OPTS



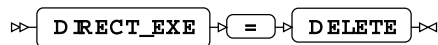
Type: String

DBGEN_NAME

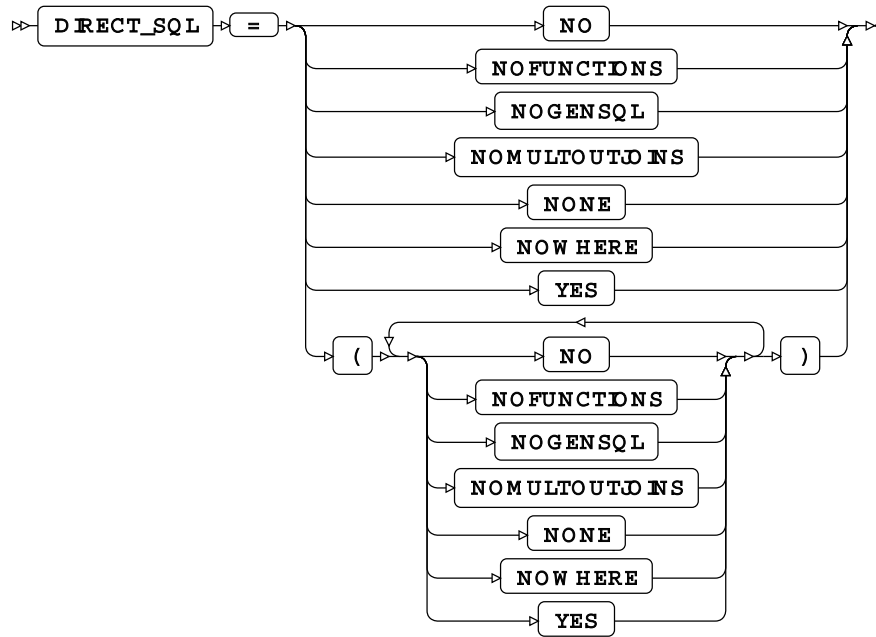


Default value: SAS

DIRECT_EXE

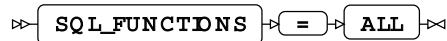


DIRECT_SQL

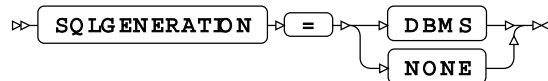


Default value: YES

SQL_FUNCTIONS

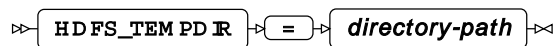


SQLGENERATION



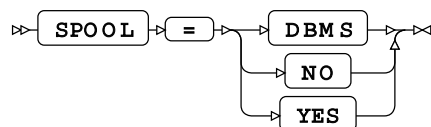
SQL transaction

HDFS_TEMPDIR



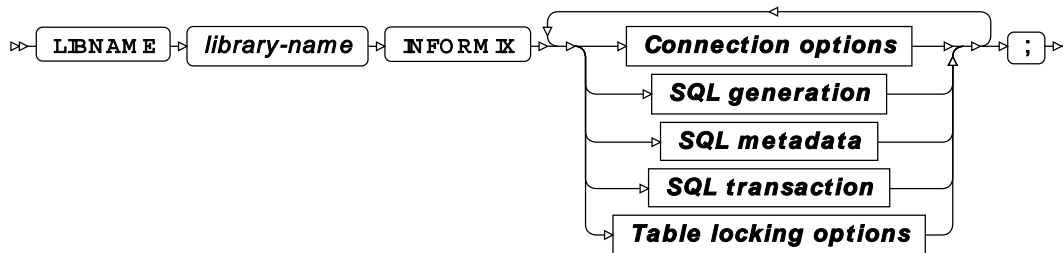
Type: String

SPOOL



WPS Engine for Informix

INFORMIX

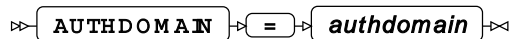


Connection options

ACCESS

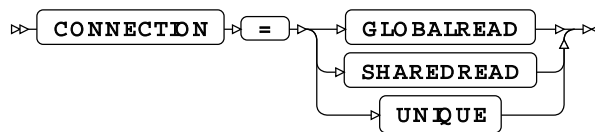


AUTHDOMAIN

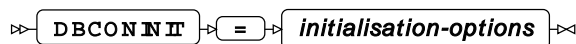


Type: String

CONNECTION

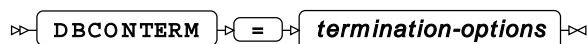


DBCONINIT



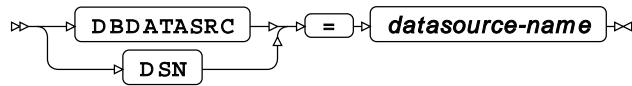
Type: String

DBCONTERM



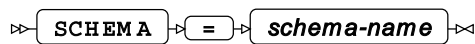
Type: String

DBDATASRC



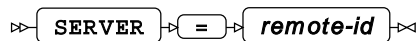
Type: String

SCHEMA



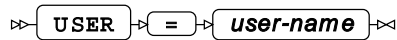
Type: String

SERVER



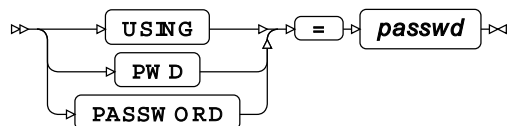
Type: String

USER



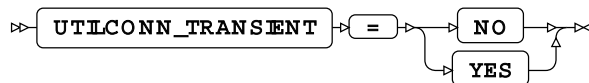
Type: String

USING



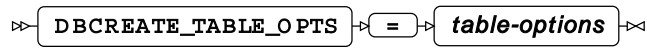
Type: String

UTILCONN_TRANSIENT



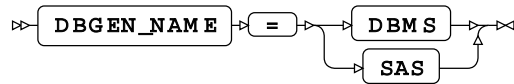
SQL generation

DBCREATE_TABLE_OPTS



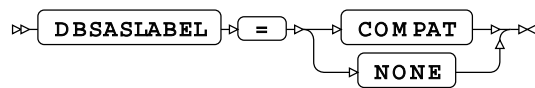
Type: String

DBGEN_NAME

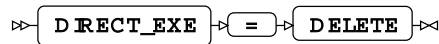


Default value: SAS

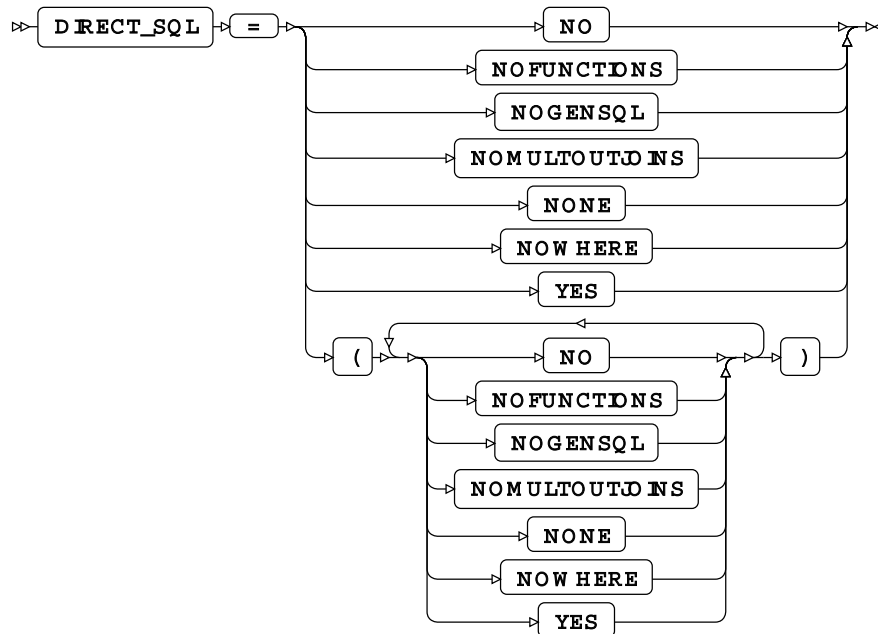
DBSASLABEL



DIRECT_EXE



DIRECT_SQL

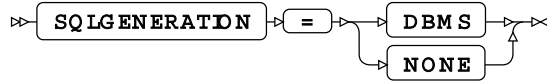


Default value: YES

SQL_FUNCTIONS



SQLGENERATION



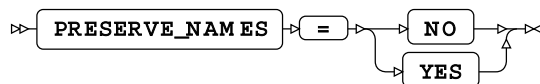
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

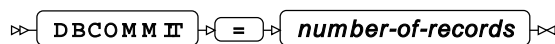
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

DBCOMMIT



Type: Numeric

Minimum value: 0

SPOOL

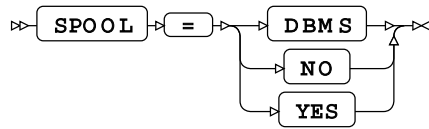
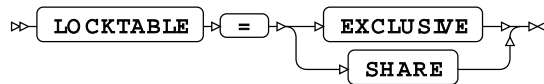
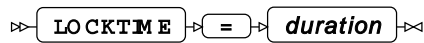


Table locking options

LOCKTABLE

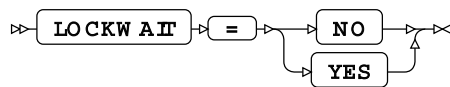


LOCKTIME

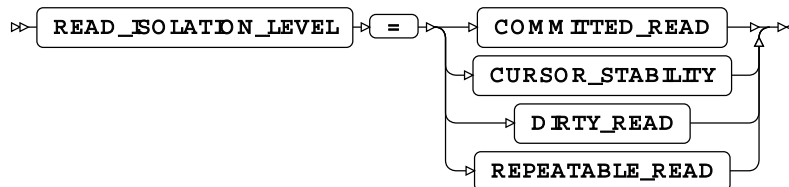


Type: Numeric

LOCKWAIT

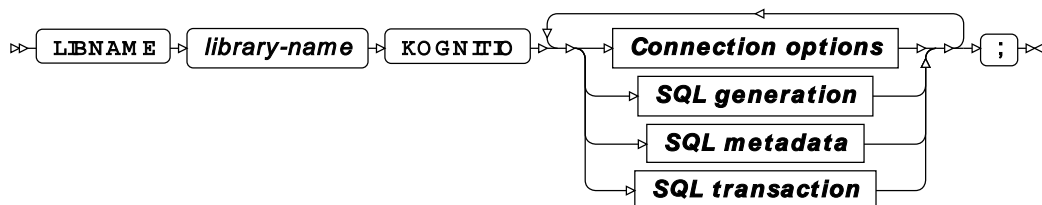


READ_ISOLATION_LEVEL



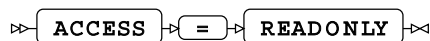
WPS Engine for Kognito

KOGNITIO



Connection options

ACCESS

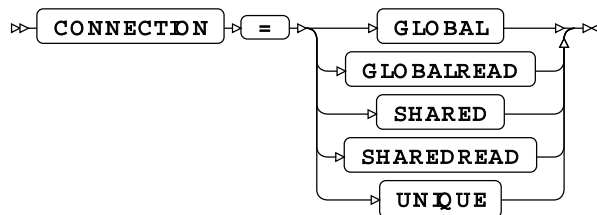


AUTHDOMAIN



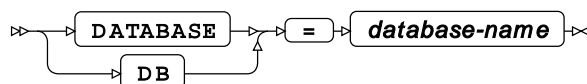
Type: String

CONNECTION



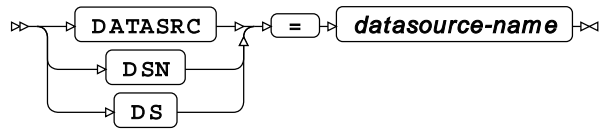
Default value: SHAREDREAD

DATABASE



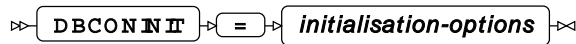
Type: String

DATASRC



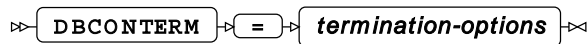
Type: String

DBCONINIT



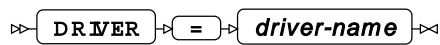
Type: String

DBCONTERM



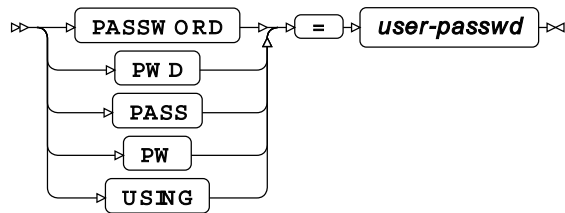
Type: String

DRIVER



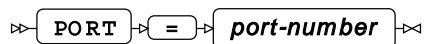
Type: String

PASSWORD



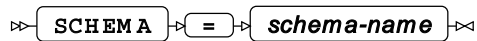
Type: String

PORT



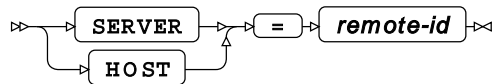
Type: String

SCHEMA



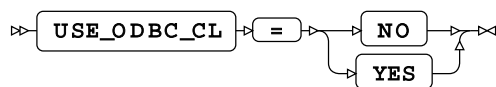
Type: String

SERVER

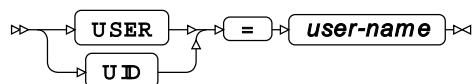


Type: String

USE_ODBC_CL

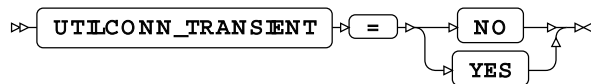


USER



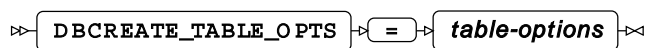
Type: String

UTILCONN_TRANSIENT



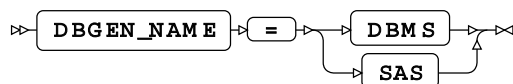
SQL generation

DBCREATE_TABLE_OPTS



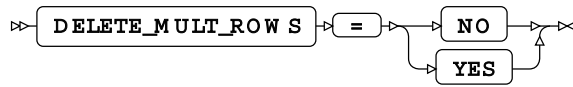
Type: String

DBGEN_NAME

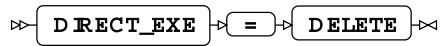


Default value: SAS

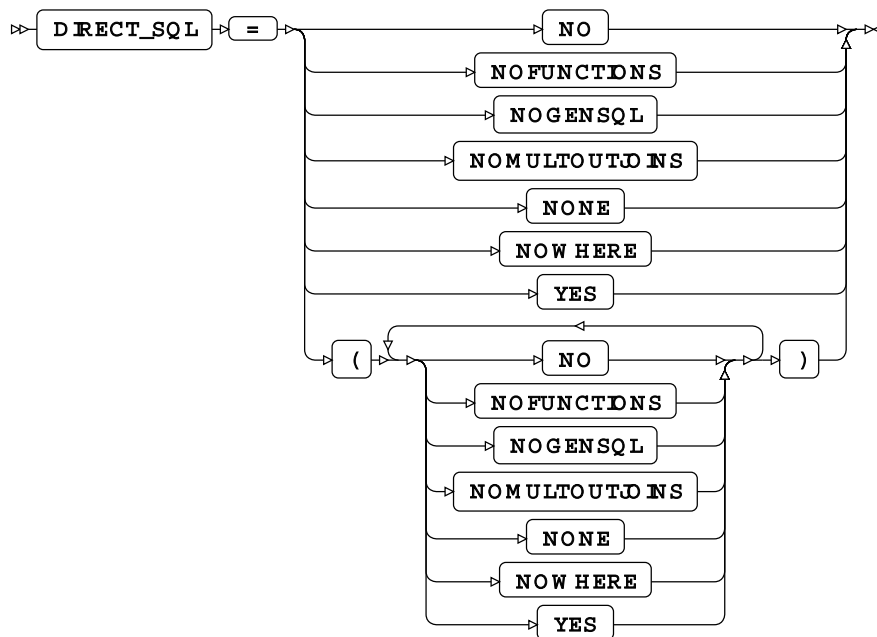
DELETE_MULT_ROWS



DIRECT_EXE

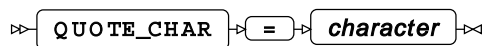


DIRECT_SQL



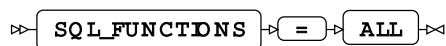
Default value: YES

QUOTE_CHAR

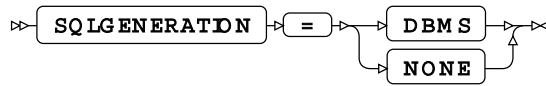


Type: String

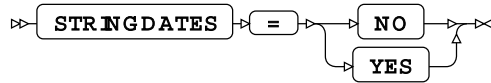
SQL_FUNCTIONS



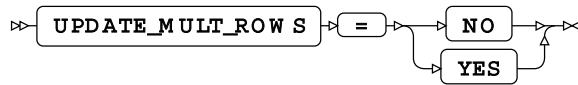
SQLGENERATION



STRINGDATES



UPDATE_MULT_ROWS



SQL metadata

IGNORE_READ_ONLY_COLUMNS

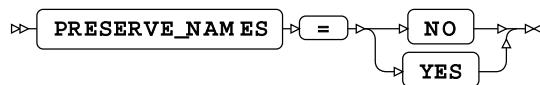


PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

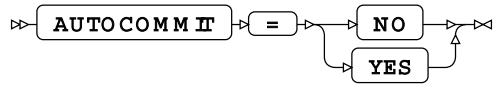
PRESERVE_TAB_NAMES



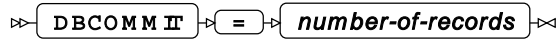
Default value: NO

SQL transaction

AUTOCOMMIT



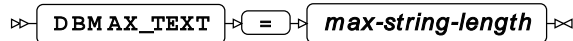
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

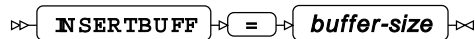


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF

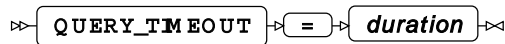


Type: Numeric

Minimum value: 1

Maximum value: 32767

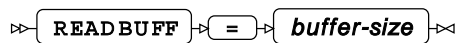
QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

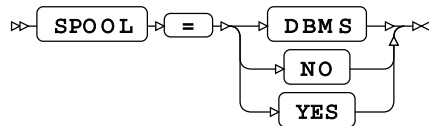


Type: Numeric

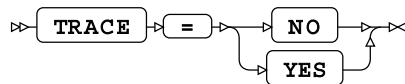
Minimum value: 1

Maximum value: 32767

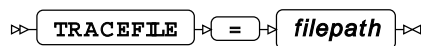
SPOOL



TRACE



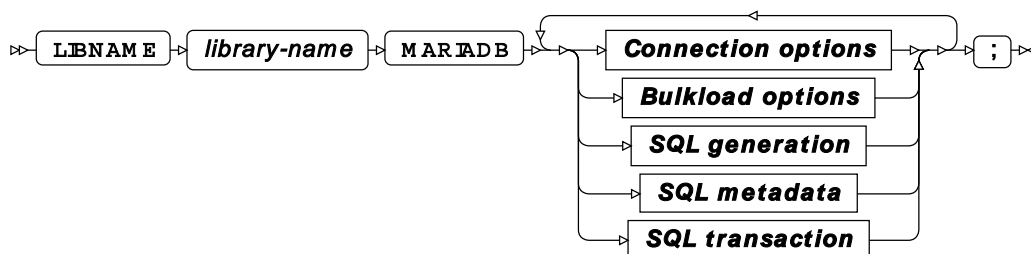
TRACEFILE



Type: String

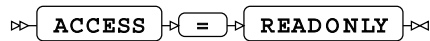
WPS Engine for MariaDB

MARIADB

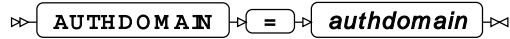


Connection options

ACCESS

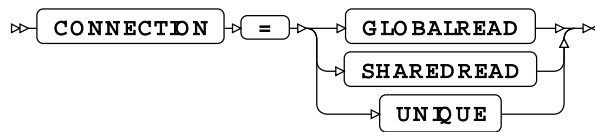


AUTHDOMAIN



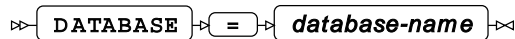
Type: String

CONNECTION



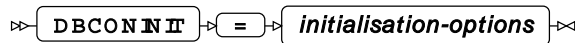
Default value: SHAREDREAD

DATABASE



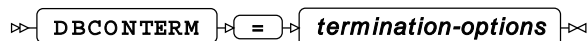
Type: String

DBCONINIT



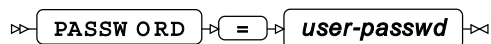
Type: String

DBCONTERM



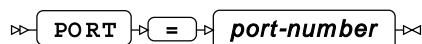
Type: String

PASSWORD



Type: String

PORT

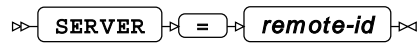


Type: Numeric

Minimum value: 0

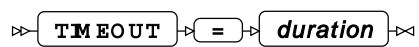
Default value: 3306

SERVER



Type: String

TIMEOUT



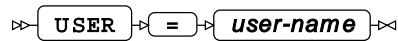
Type: Numeric

Minimum value: 1

Maximum value: 1000

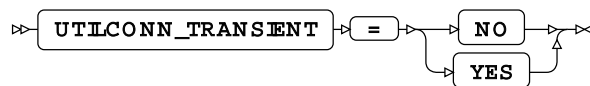
Default value: 15

USER



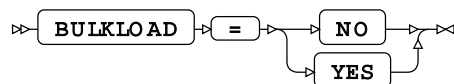
Type: String

UTILCONN_TRANSIENT



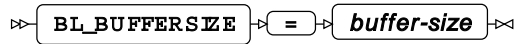
Bulkload options

BULKLOAD



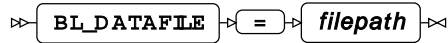
Default value: NO

BL_BUFFERSIZE



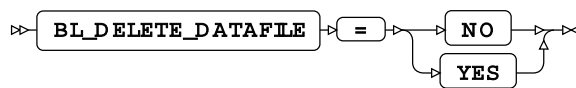
Type: String

BL_DATAFILE



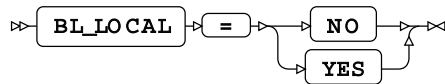
Type: String

BL_DELETE_DATAFILE



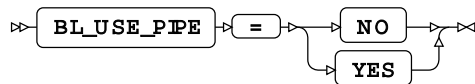
Default value: YES

BL_LOCAL



Default value: YES

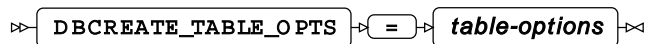
BL_USE_PIPE



Default value: NO

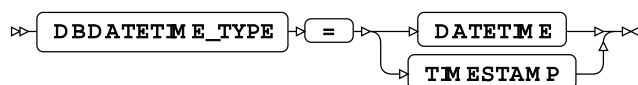
SQL generation

DBCREATE_TABLE_OPTS



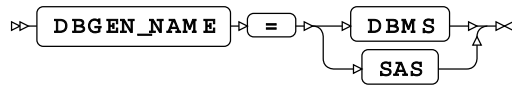
Type: String

DBDATETIME_TYPE



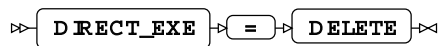
Default value: `TIMESTAMP`

DBGEN_NAME

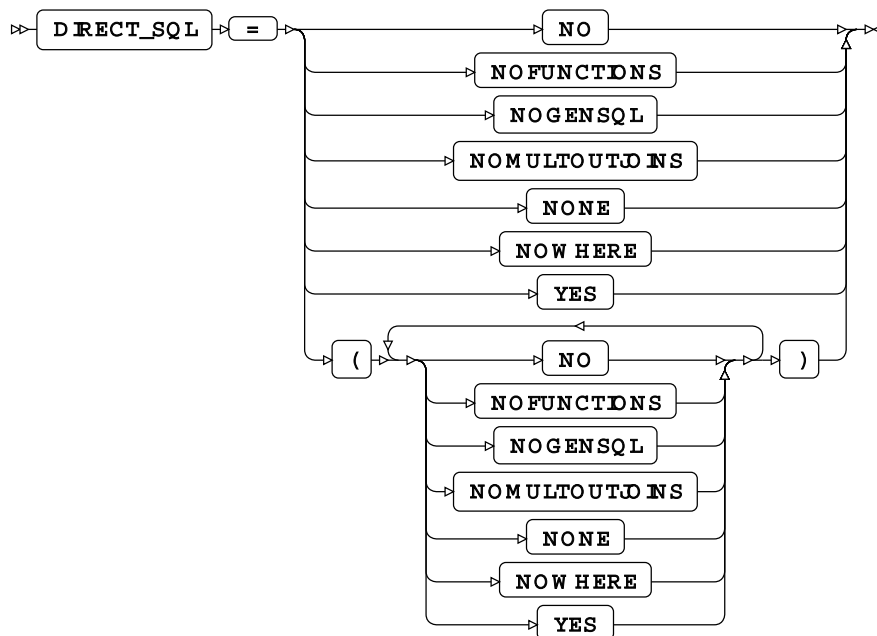


Default value: `SAS`

DIRECT_EXE

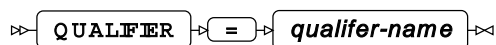


DIRECT_SQL



Default value: `YES`

QUALIFIER

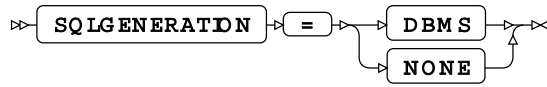


Type: String

SQL_FUNCTIONS



SQLGENERATION



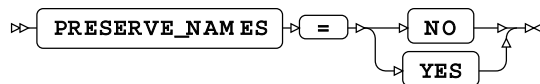
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

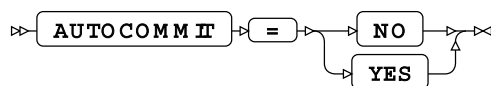
PRESERVE_TAB_NAMES



Default value: NO

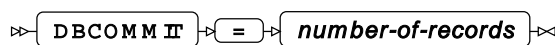
SQL transaction

AUTOCOMMIT



Default value: NO

DBCOMMIT

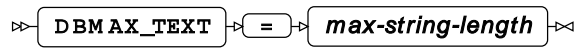


Type: Numeric

Minimum value: 0

Default value: 1000

DBMAX_TEXT



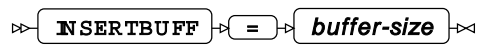
Type: Numeric

Minimum value: 1

Maximum value: 32767

Default value: 1024

INSERTBUFF

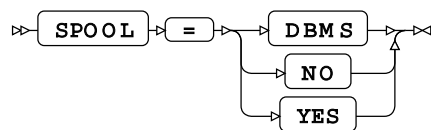


Type: Numeric

Minimum value: 1

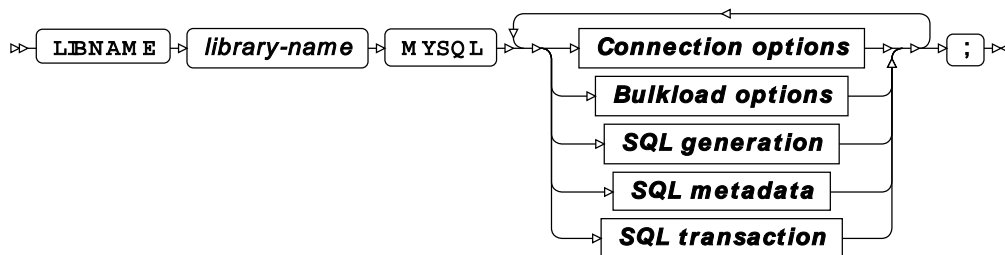
Maximum value: 32767

SPOOL



WPS Engine for MySQL

MYSQL

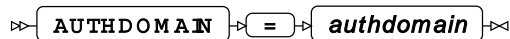


Connection options

ACCESS

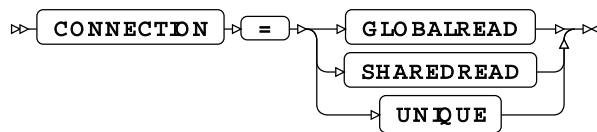


AUTHDOMAIN



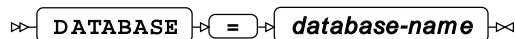
Type: String

CONNECTION



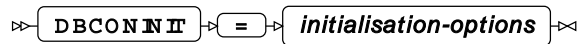
Default value: SHAREDREAD

DATABASE



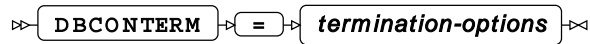
Type: String

DBCONINIT



Type: String

DBCONTERM



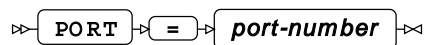
Type: String

PASSWORD



Type: String

PORT

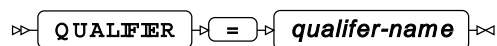


Type: Numeric

Minimum value: 0

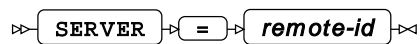
Default value: 3306

QUALIFIER



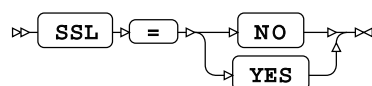
Type: String

SERVER



Type: String

SSL



Default value: FALSE

SSL_CA

```
graph LR; A[SSL_CA] --> B[=]; B --> C[certificate-authority-file];
```

Type: String**SSL_CAPATH**

```
graph LR; A[SSL_CAPATH] --> B[=]; B --> C[certificate-authority-directory];
```

Type: String**SSL_CERT**

```
graph LR; A[SSL_CERT] --> B[=]; B --> C[certificate-file];
```

Type: String**SSL_CIPHER**

```
graph LR; A[SSL_CIPHER] --> B[=]; B --> C[permissible-ssl-ciphers];
```

Type: String**SSL_KEY**

```
graph LR; A[SSL_KEY] --> B[=]; B --> C[key-file];
```

Type: String**TIMEOUT**

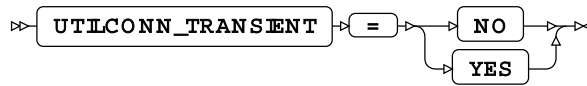
```
graph LR; A[TIMEOUT] --> B[=]; B --> C[duration];
```

Type: Numeric**Minimum value:** 1**Maximum value:** 1000**Default value:** 15**USER**

```
graph LR; A[USER] --> B[=]; B --> C[user-name];
```

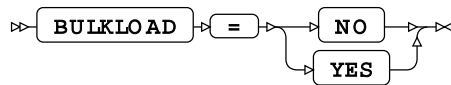
Type: String

UTILCONN_TRANSIENT



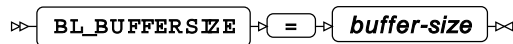
Bulkload options

BULKLOAD



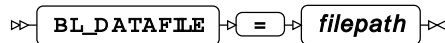
Default value: NO

BL_BUFFERSIZE



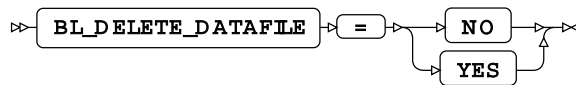
Type: String

BL_DATAFILE



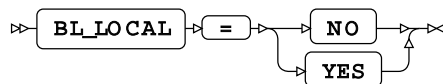
Type: String

BL_DELETE_DATAFILE



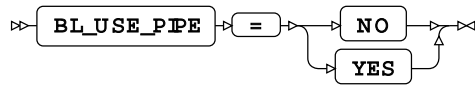
Default value: YES

BL_LOCAL



Default value: YES

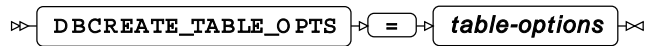
BL_USE_PIPE



Default value: NO

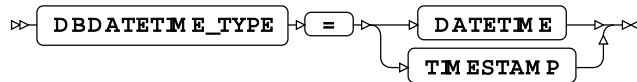
SQL generation

DBCREATE_TABLE_OPTS



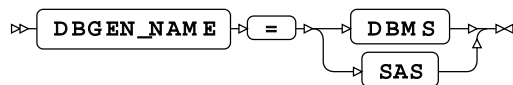
Type: String

DBDATETIME_TYPE



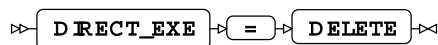
Default value: TIMESTAMP

DBGEN_NAME

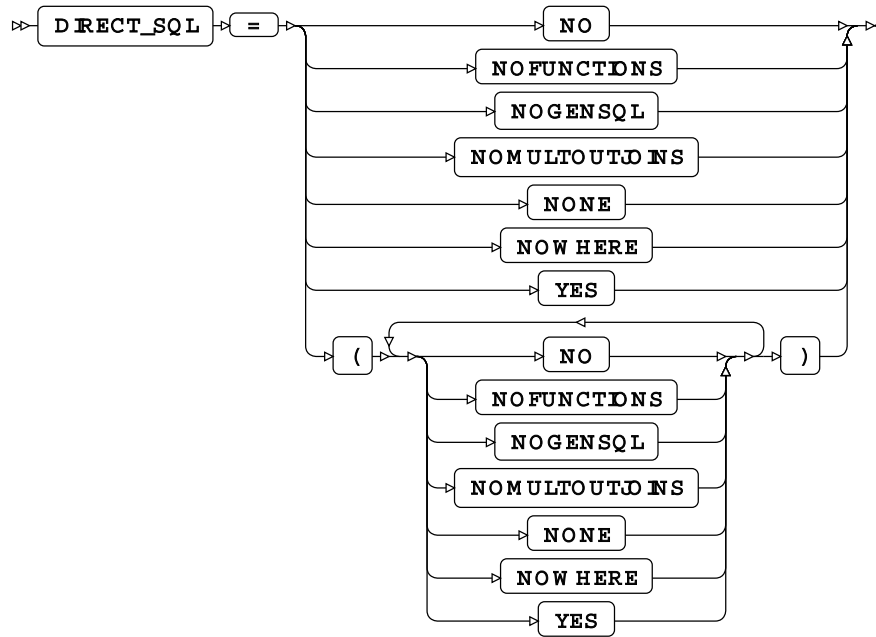


Default value: SAS

DIRECT_EXE

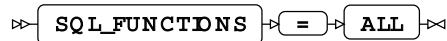


DIRECT_SQL

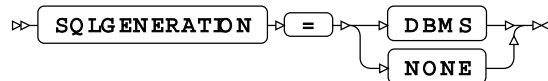


Default value: YES

SQL_FUNCTIONS



SQLGENERATION



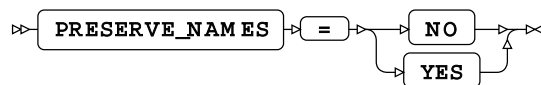
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

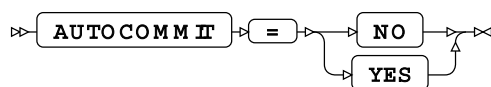
PRESERVE_TAB_NAMES



Default value: NO

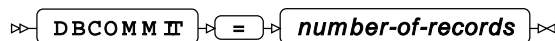
SQL transaction

AUTOCOMMIT



Default value: NO

DBCOMMIT

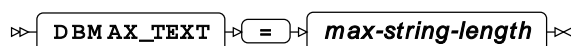


Type: Numeric

Minimum value: 0

Default value: 1000

DBMAX_TEXT



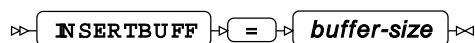
Type: Numeric

Minimum value: 1

Maximum value: 32767

Default value: 1024

INSERTBUFF

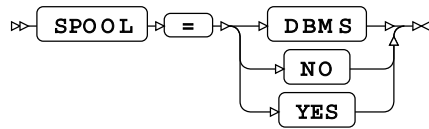


Type: Numeric

Minimum value: 1

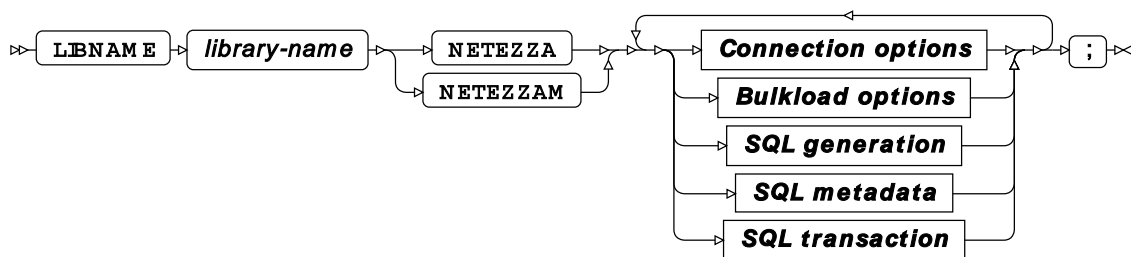
Maximum value: 32767

SPOOL



WPS Engine for Netezza

NETEZZA

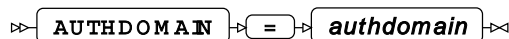


Connection options

ACCESS

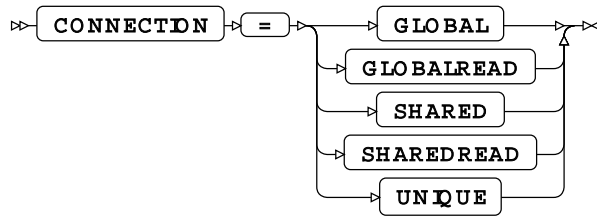


AUTHDOMAIN

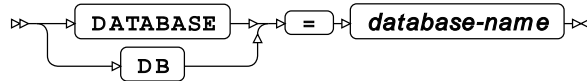


Type: String

CONNECTION

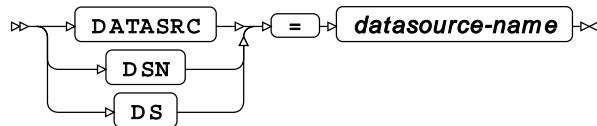


DATABASE



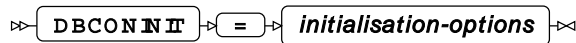
Type: String

DATASRC



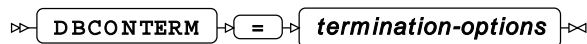
Type: String

DBCONINIT



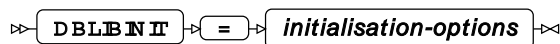
Type: String

DBCONTERM



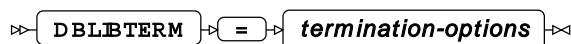
Type: String

DBLIBINIT



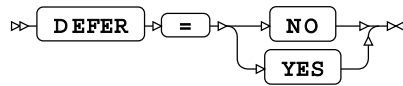
Type: String

DBLIBTERM

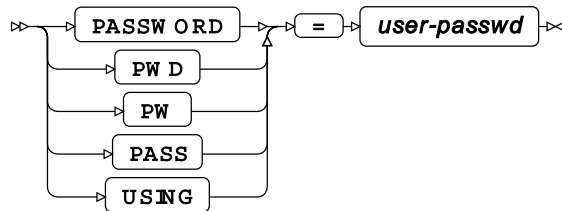


Type: String

DEFER

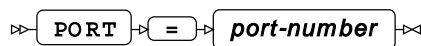


PASSWORD



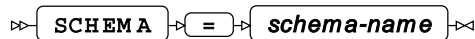
Type: String

PORT



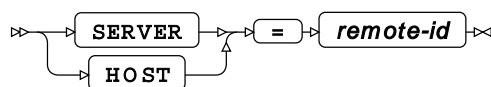
Type: Numeric

SCHEMA



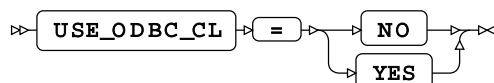
Type: String

SERVER

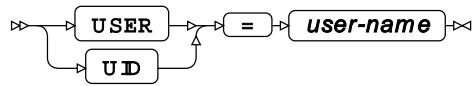


Type: String

USE_ODBC_CL



USER



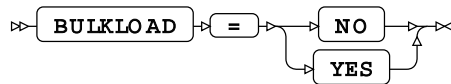
Type: String

UTILCONN_TRANSIENT

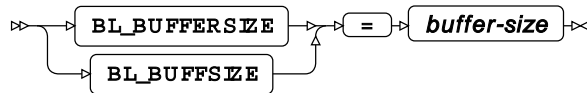


Bulkload options

BULKLOAD

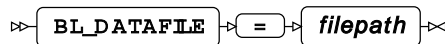


BL_BUFFERSIZE



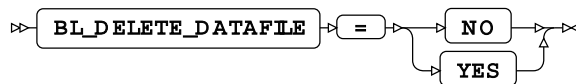
Type: String

BL_DATAFILE



Type: String

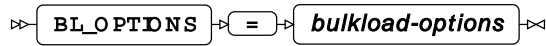
BL_DELETE_DATAFILE



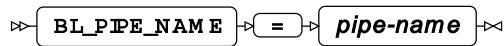
BL_DELIMITER



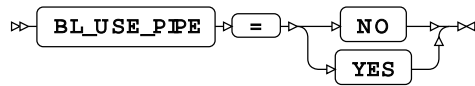
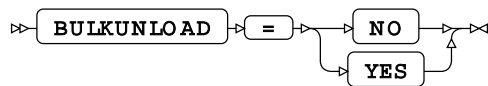
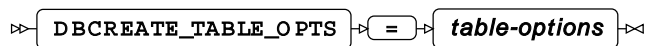
Type: String

BL_OPTIONS

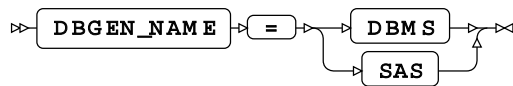
Type: String

BL_PIPE_NAME

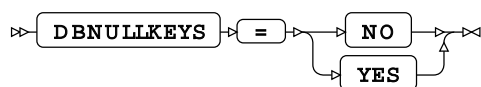
Type: String

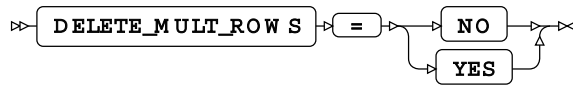
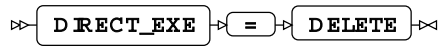
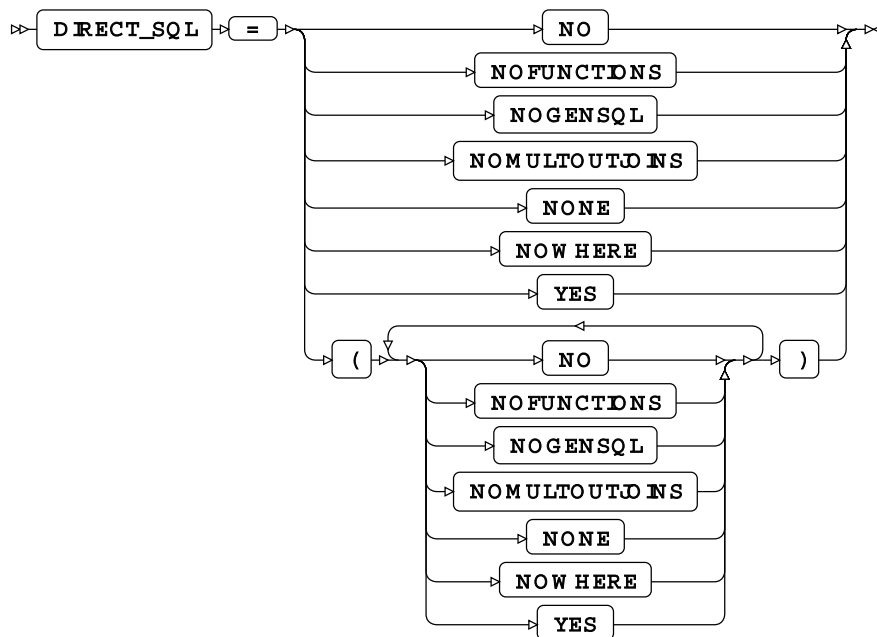
BL_USE_PIPE**BULKUNLOAD****SQL generation****DBCREATE_TABLE_OPTS**

Type: String

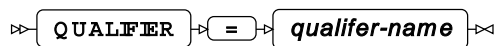
DBGEN_NAME

Default value: SAS

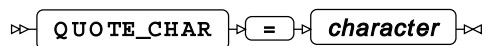
DBNULLKEYS

DELETE_MULT_ROWS**DIRECT_EXE****DIRECT_SQL**

Default value: YES

QUALIFIER

Type: String

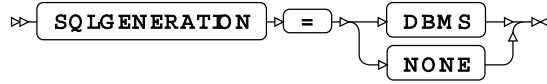
QUOTE_CHAR

Type: String

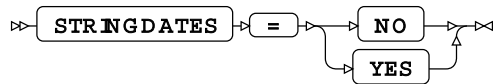
SQL_FUNCTIONS



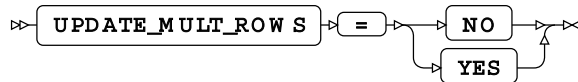
SQLGENERATION



STRINGDATES



UPDATE_MULT_ROWS



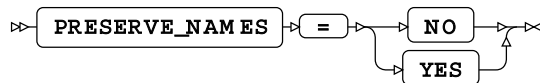
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

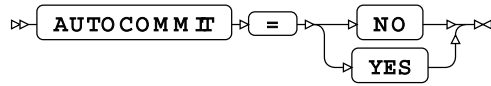
PRESERVE_TAB_NAMES



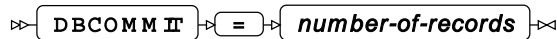
Default value: NO

SQL transaction

AUTOCOMMIT



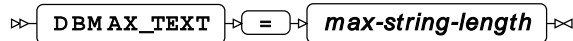
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

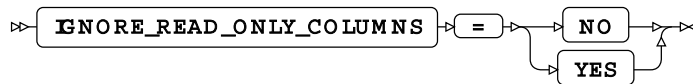


Type: Numeric

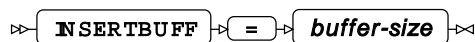
Minimum value: 1

Maximum value: 32767

IGNORE_READ_ONLY_COLUMNS



INSERTBUFF

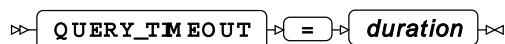


Type: Numeric

Minimum value: 1

Maximum value: 32767

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

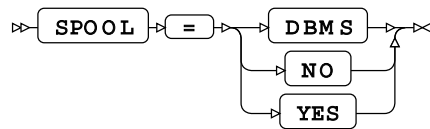


Type: Numeric

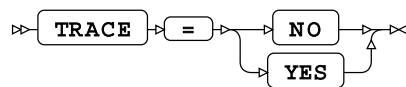
Minimum value: 1

Maximum value: 32767

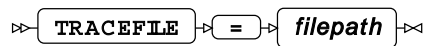
SPOOL



TRACE



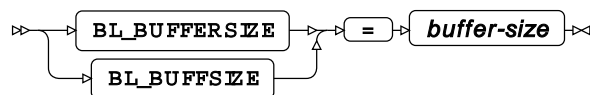
TRACEFILE



Type: String

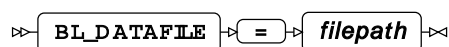
NETEZZA Dataset Options

BL_BUFFERSIZE



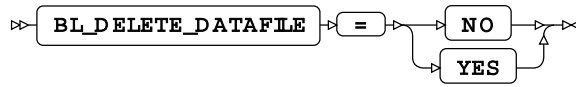
Type: String

BL_DATAFILE

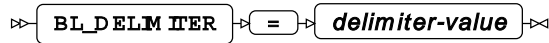


Type: String

BL_DELETE_DATAFILE

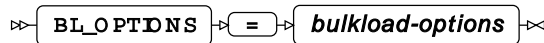


BL_DELIMITER



Type: String

BL_OPTIONS



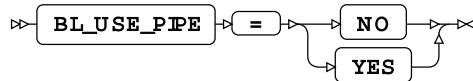
Type: String

BL_PIPE_NAME

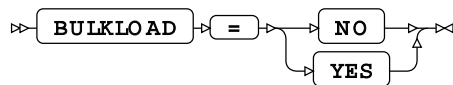


Type: String

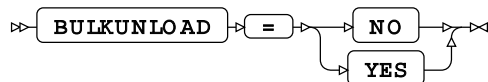
BL_USE_PIPE



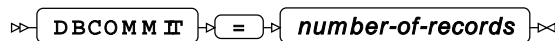
BULKLOAD



BULKUNLOAD



DBCMMIT



Type: Numeric

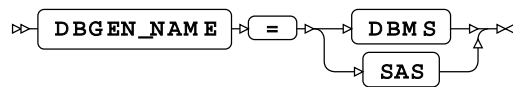
Minimum value: 0

DBCREATE_TABLE_OPTS

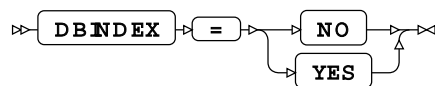


Type: String

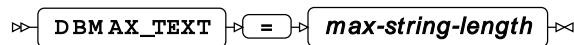
DBGEN_NAME



DBINDEX

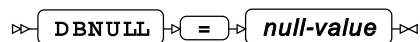


DBMAX_TEXT

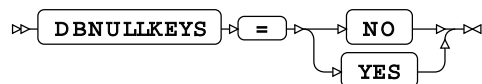


Type: Numeric

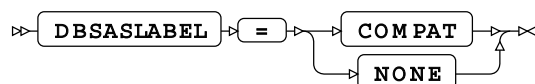
DBNULL



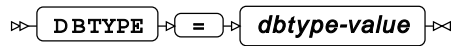
DBNULLKEYS



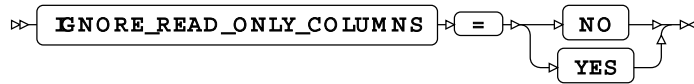
DBSASLABEL



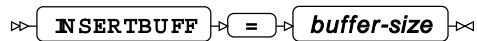
DBTYPE



IGNORE_READ_ONLY_COLUMNS



INSERTBUFF



Type: Numeric

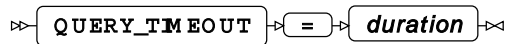
Minimum value: 1

PRESERVE_COL_NAMES



Default value: FALSE

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

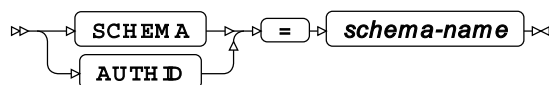


Type: Numeric

Minimum value: 1

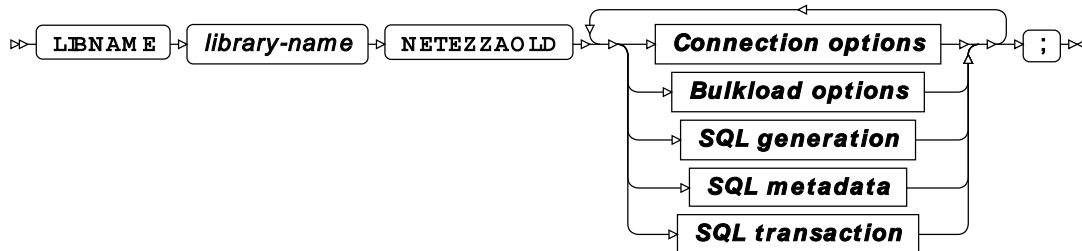
Maximum value: 32767

SCHEMA



Type: String

NETEZZAOLD

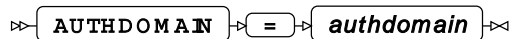


Connection options

ACCESS

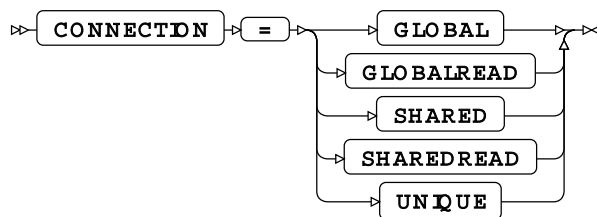


AUTHDOMAIN

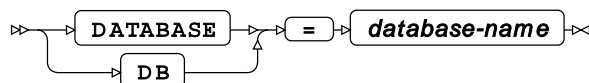


Type: String

CONNECTION

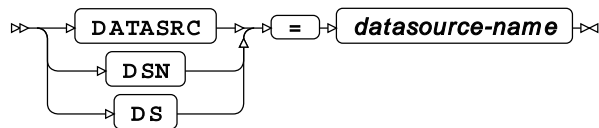


DATABASE



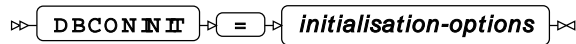
Type: String

DATASRC



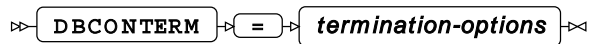
Type: String

DBCONINIT



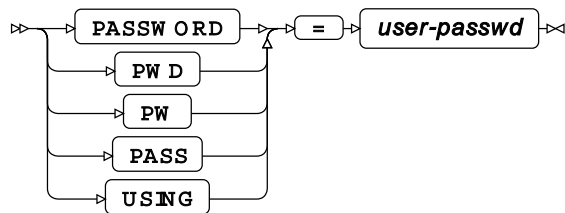
Type: String

DBCONTERM



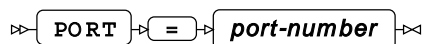
Type: String

PASSWORD



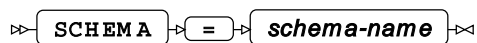
Type: String

PORT



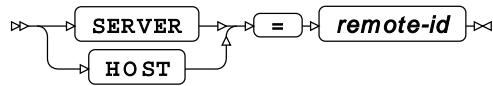
Type: String

SCHEMA



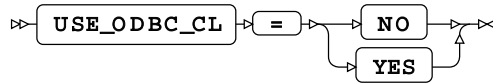
Type: String

SERVER

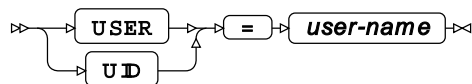


Type: String

USE_ODBC_CL

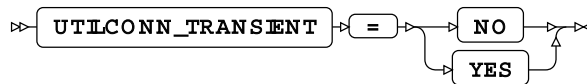


USER



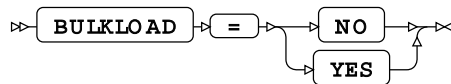
Type: String

UTILCONN_TRANSIENT

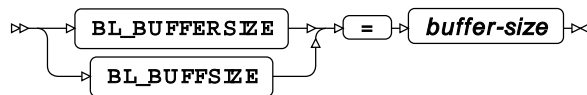


Bulkload options

BULKLOAD

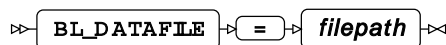


BL_BUFFERSIZE



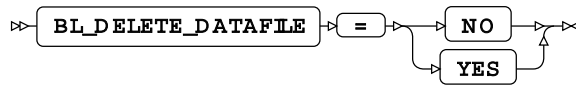
Type: String

BL_DATAFILE

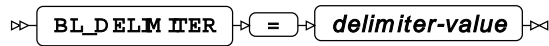


Type: String

BL_DELETE_DATAFILE

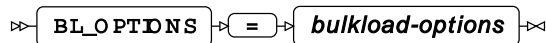


BL_DELIMITER



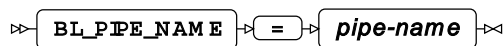
Type: String

BL_OPTIONS



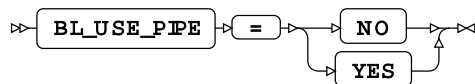
Type: String

BL_PIPE_NAME

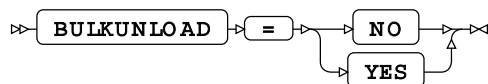


Type: String

BL_USE_PIPE

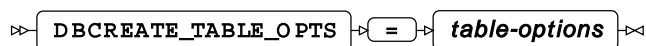


BULKUNLOAD



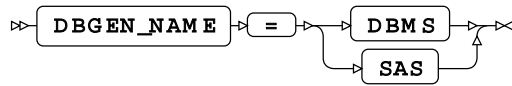
SQL generation

DBCREATE_TABLE_OPTS



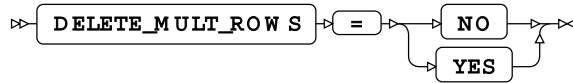
Type: String

DBGEN_NAME

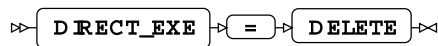


Default value: SAS

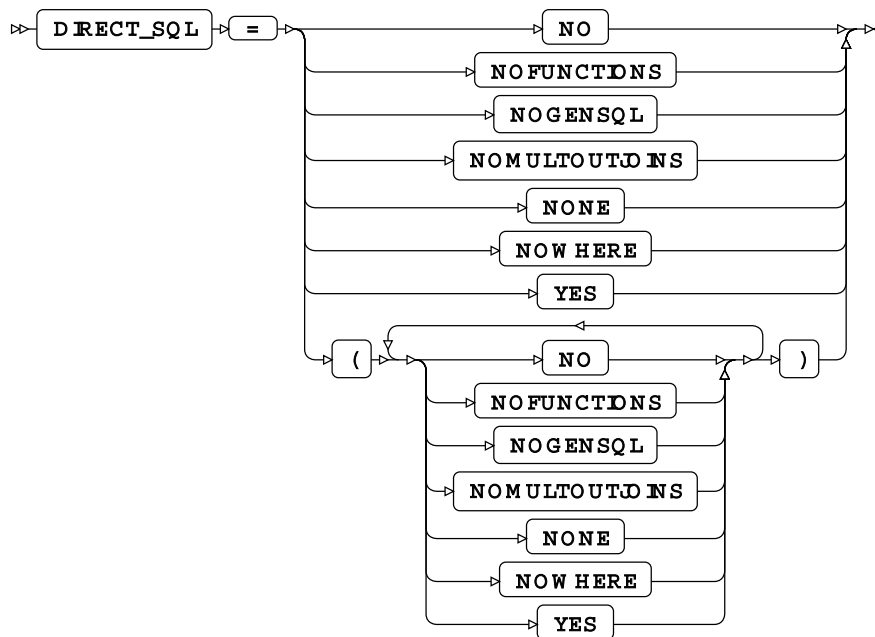
DELETE_MULT_ROWS



DIRECT_EXE

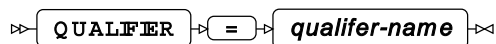


DIRECT_SQL



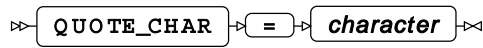
Default value: YES

QUALIFIER



Type: String

QUOTE_CHAR

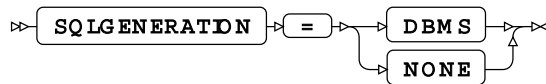


Type: String

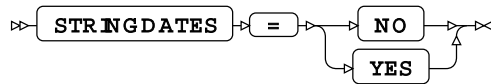
SQL_FUNCTIONS



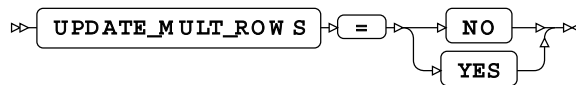
SQLGENERATION



STRINGDATES



UPDATE_MULT_ROWS



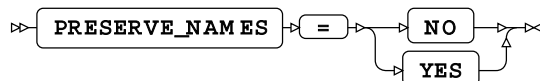
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

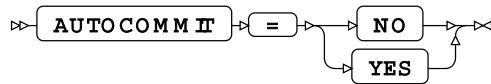
PRESERVE_TAB_NAMES



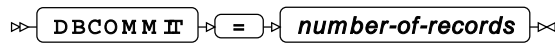
Default value: NO

SQL transaction

AUTOCOMMIT



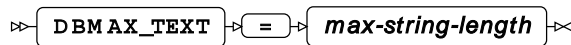
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

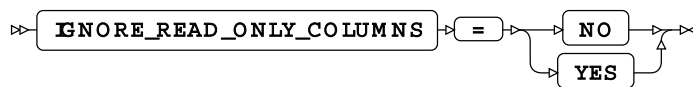


Type: Numeric

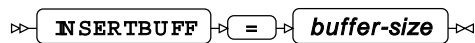
Minimum value: 1

Maximum value: 32767

IGNORE_READ_ONLY_COLUMNS



INSERTBUFF

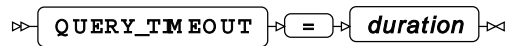


Type: Numeric

Minimum value: 1

Maximum value: 32767

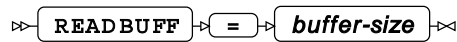
QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

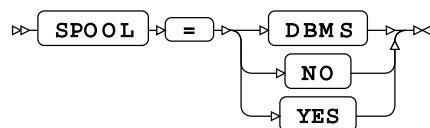


Type: Numeric

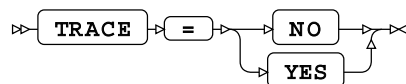
Minimum value: 1

Maximum value: 32767

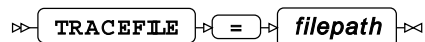
SPOOL



TRACE



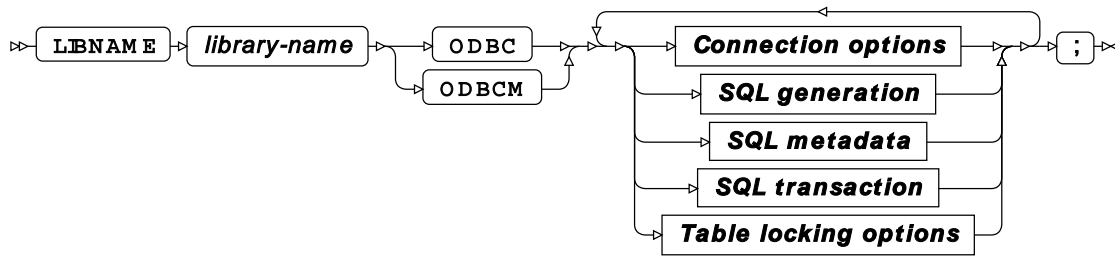
TRACEFILE



Type: String

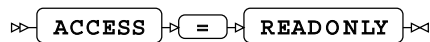
WPS Engine for ODBC

ODBC

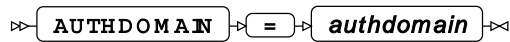


Connection options

ACCESS

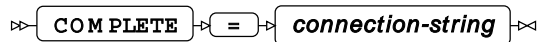


AUTHDOMAIN



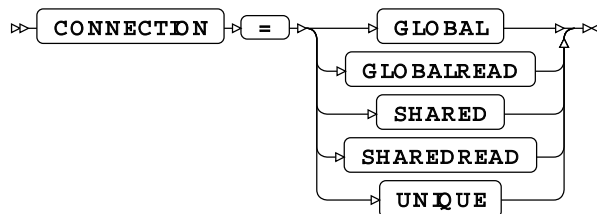
Type: String

COMPLETE

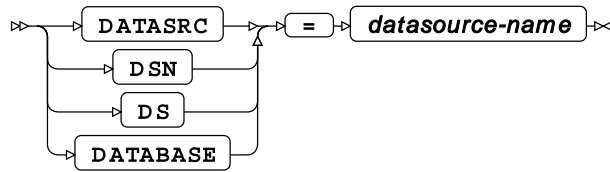


Type: String

CONNECTION



DATASRC



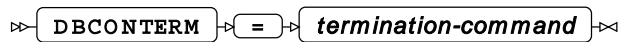
Type: String

DBCONINIT



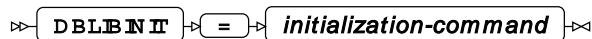
Type: String

DBCONTERM



Type: String

DBLIBINIT



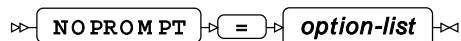
Type: String

DBLIBTERM



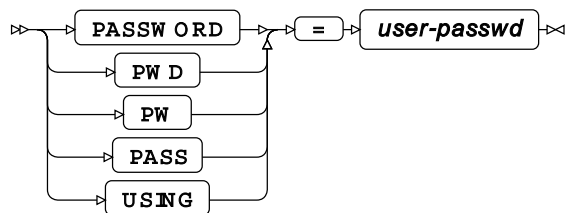
Type: String

NOPROMPT



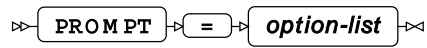
Type: String

PASSWORD



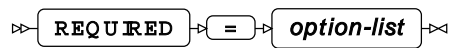
Type: String

PROMPT



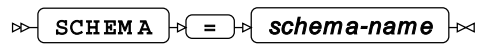
Type: String

REQUIRED



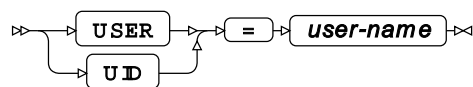
Type: String

SCHEMA



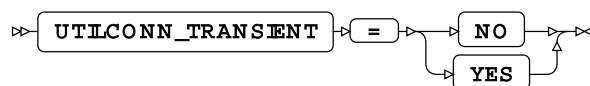
Type: String

USER



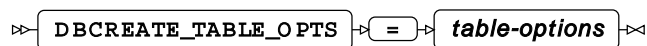
Type: String

UTILCONN_TRANSIENT



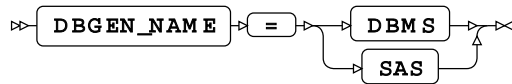
SQL generation

DBCREATE_TABLE_OPTS

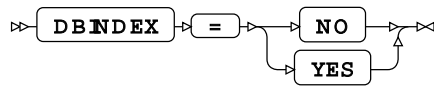


Type: String

DBGEN_NAME

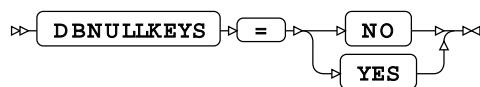


DBINDEX

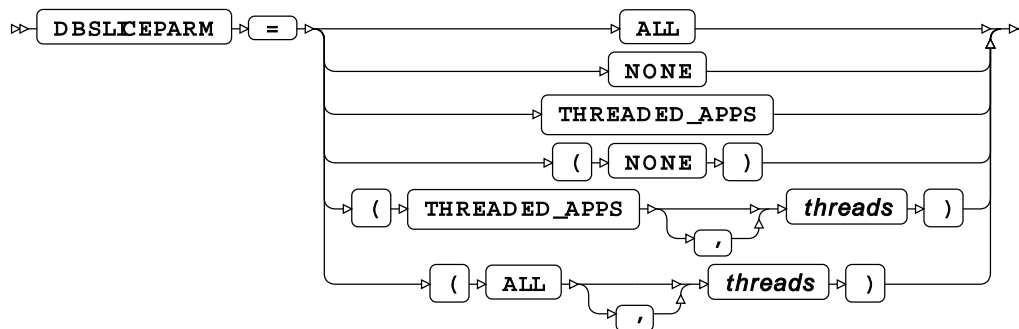


Default value: YES

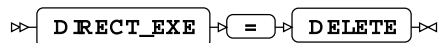
DBNULLKEYS



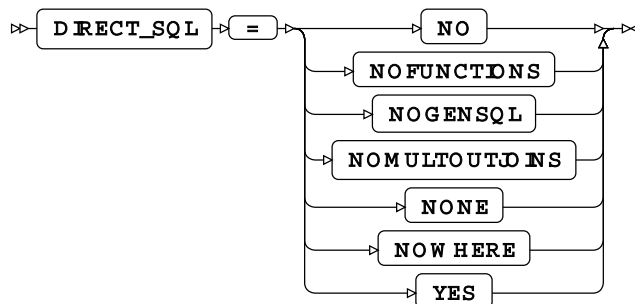
DBSLICEPARM



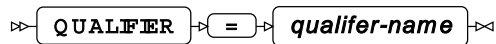
DIRECT_EXE



DIRECT_SQL

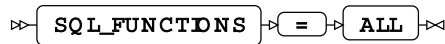


QUALIFIER

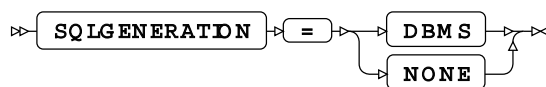


Type: String

SQL_FUNCTIONS



SQLGENERATION



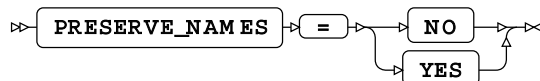
Default value: NONE

SQL metadata

PRESERVE_COL_NAMES



PRESERVE_NAMES

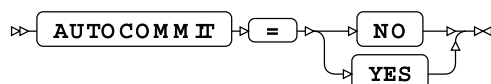


PRESERVE_TAB_NAMES

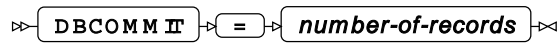


SQL transaction

AUTOCOMMIT



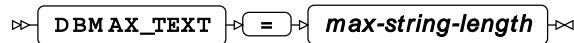
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



Type: Numeric

Minimum value: 1

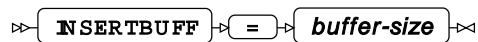
Maximum value: 32767

IGNORE_READ_ONLY_COLUMNS



Default value: NO

INSERTBUFF



Type: Numeric

Minimum value: 1

Maximum value: 32767

READBUFF



Type: Numeric

Minimum value: 1

Maximum value: 32767

SPOOL

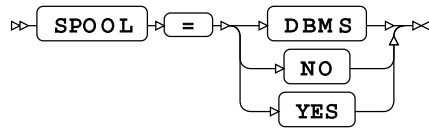
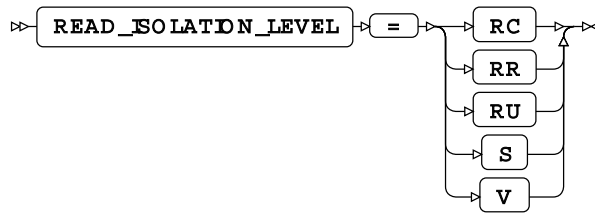
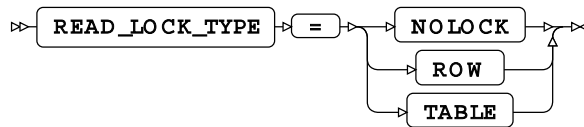


Table locking options

READ_ISOLATION_LEVEL

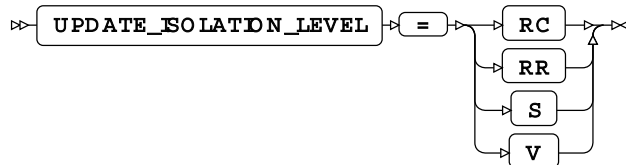


READ_LOCK_TYPE

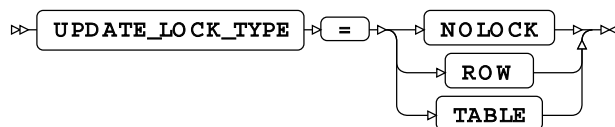


Default value: NOLOCK

UPDATE_ISOLATION_LEVEL



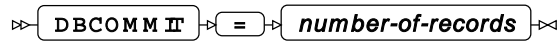
UPDATE_LOCK_TYPE



Default value: NOLOCK

ODBC Dataset Options

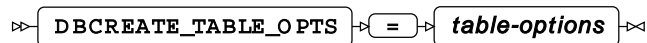
DBCMMIT



Type: Numeric

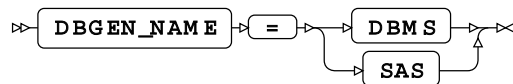
Minimum value: 0

DBCREATE_TABLE_OPTS

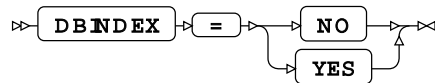


Type: String

DBGEN_NAME

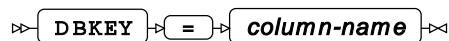


DBINDEX

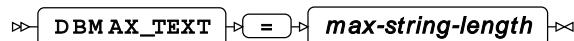


Default value: TRUE

DBKEY

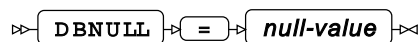


DBMAX_TEXT

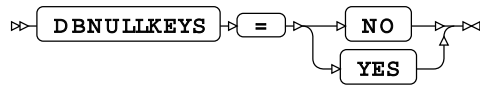


Type: Numeric

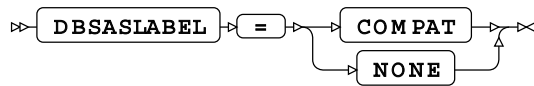
DBNULL



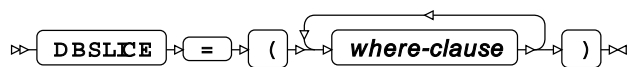
DBNULLKEYS



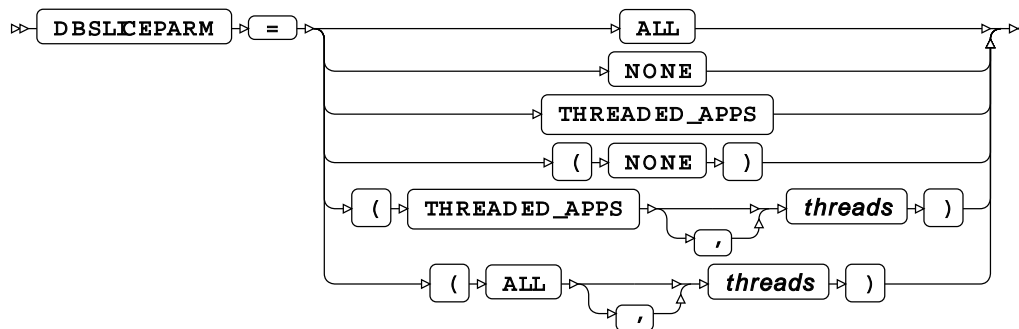
DBSASLABEL



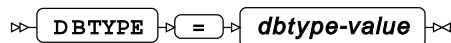
DBSLICE



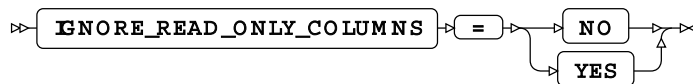
DBSLICEPARM



DBTYPE

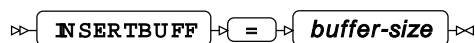


IGNORE_READ_ONLY_COLUMNS



Default value: FALSE

INSERTBUFF



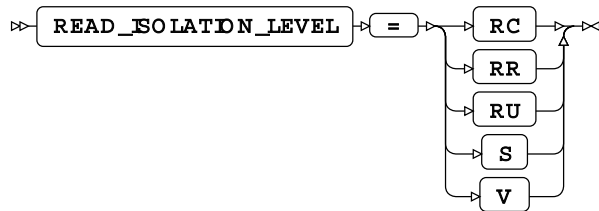
Type: Numeric

Minimum value: 1

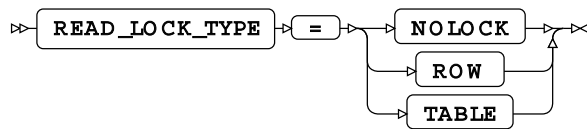
PRESERVE_COL_NAMES



READ_ISOLATION_LEVEL

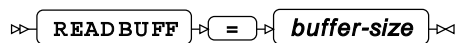


READ_LOCK_TYPE



Default value: NOLOCK

READBUFF

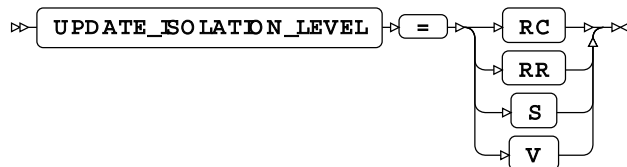


Type: Numeric

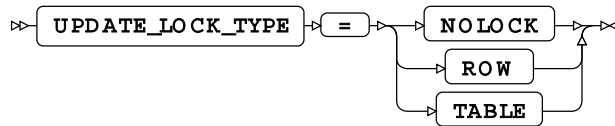
Minimum value: 1

Maximum value: 32767

UPDATE_ISOLATION_LEVEL

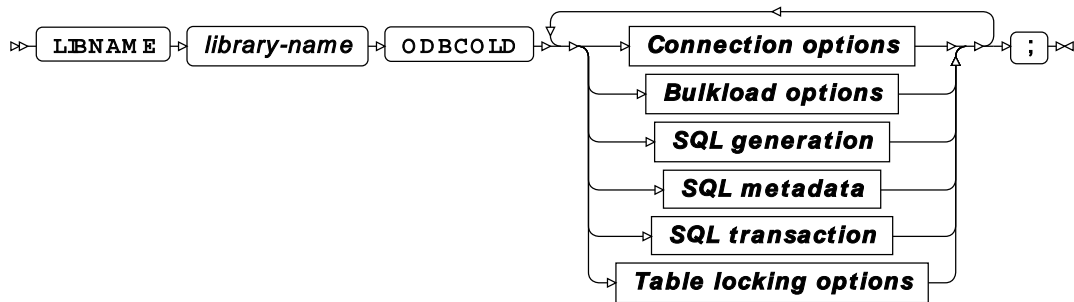


UPDATE_LOCK_TYPE



Default value: NOLOCK

ODBCOLD

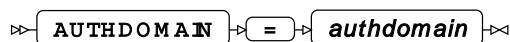


Connection options

ACCESS

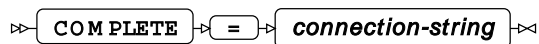


AUTHDOMAIN



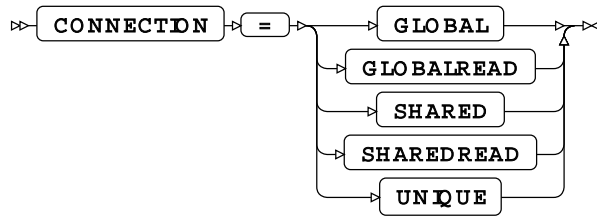
Type: String

COMPLETE

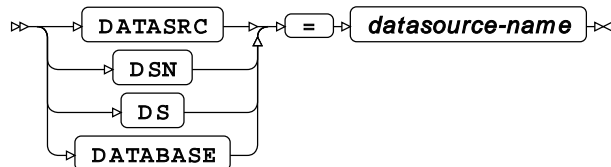


Type: String

CONNECTION

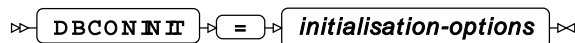


DATASRC



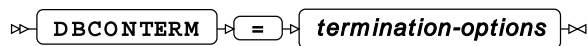
Type: String

DBCONINIT



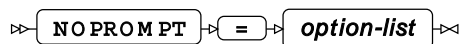
Type: String

DBCONTERM



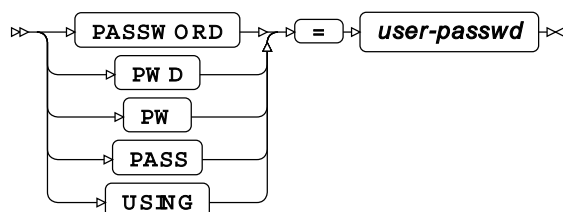
Type: String

NOPROMPT



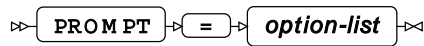
Type: String

PASSWORD



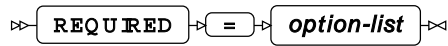
Type: String

PROMPT



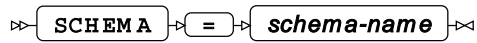
Type: String

REQUIRED



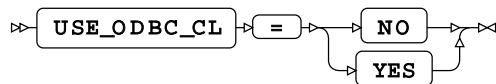
Type: String

SCHEMA

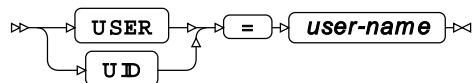


Type: String

USE_ODBC_CL

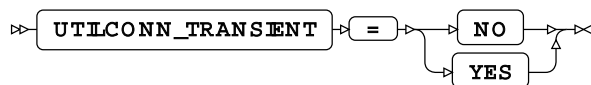


USER



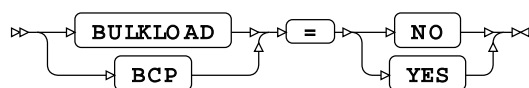
Type: String

UTILCONN_TRANSIENT



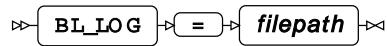
Bulkload options

BULKLOAD



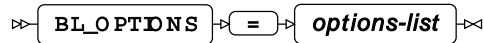
Default value: NO

BL_LOG



Type: String

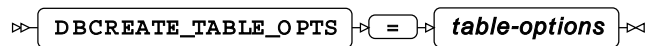
BL_OPTIONS



Type: String

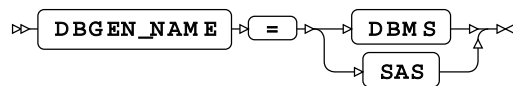
SQL generation

DBCREATE_TABLE_OPTS



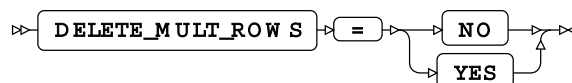
Type: String

DBGEN_NAME

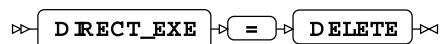


Default value: SAS

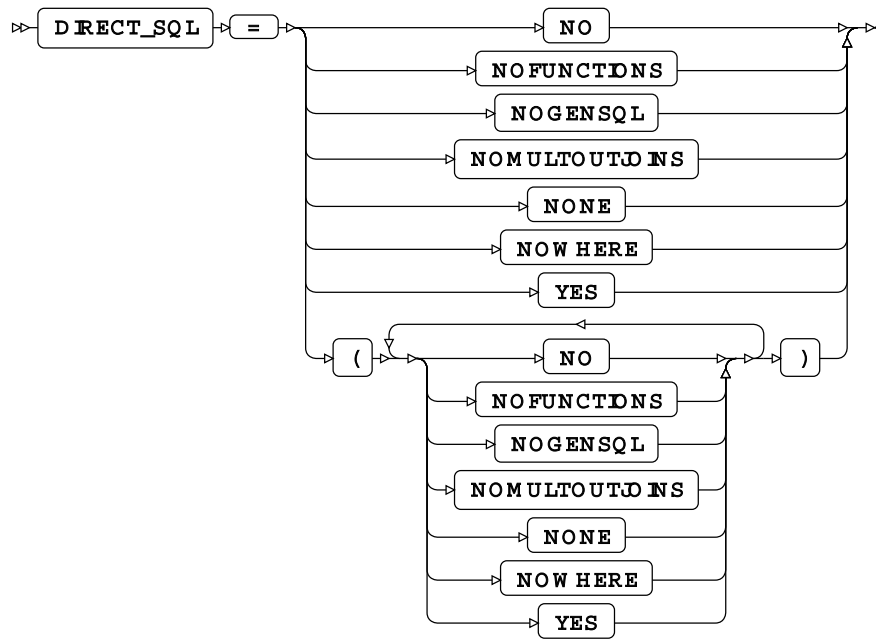
DELETE_MULT_ROWS



DIRECT_EXE

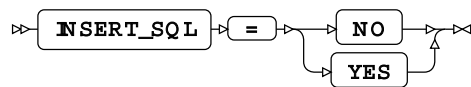


DIRECT_SQL

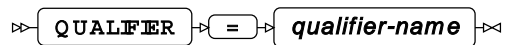


Default value: YES

INSERT_SQL

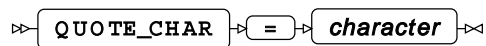


QUALIFIER



Type: String

QUOTE_CHAR

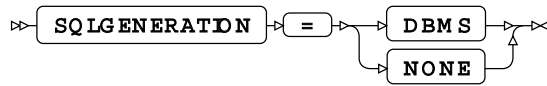


Type: String

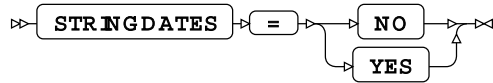
SQL_FUNCTIONS



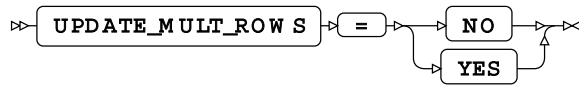
SQLGENERATION



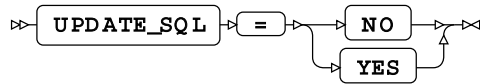
STRINGDATES



UPDATE_MULT_ROWS

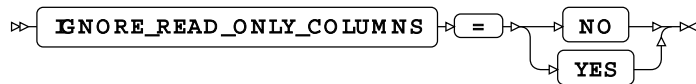


UPDATE_SQL



SQL metadata

IGNORE_READ_ONLY_COLUMNS



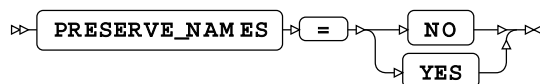
Default value: NO

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

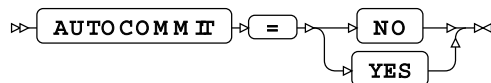
PRESERVE_TAB_NAMES



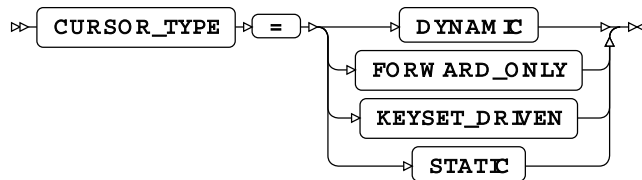
Default value: NO

SQL transaction

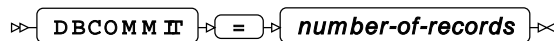
AUTOCOMMIT



CURSOR_TYPE



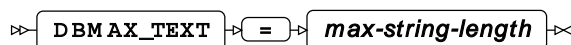
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



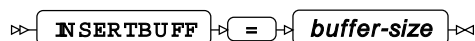
Type: Numeric

Minimum value: 1

Maximum value: 32767

Default value: 4000

INSERTBUFF

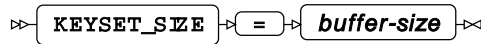


Type: Numeric

Minimum value: 1

Maximum value: 32767

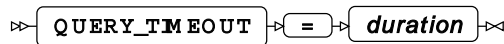
KEYSET_SIZE



Type: Numeric

Minimum value: 0

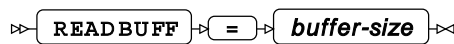
QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

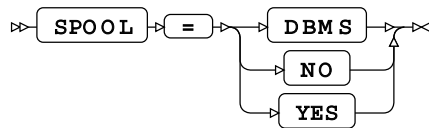


Type: Numeric

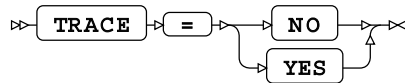
Minimum value: 1

Maximum value: 32767

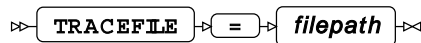
SPOOL



TRACE



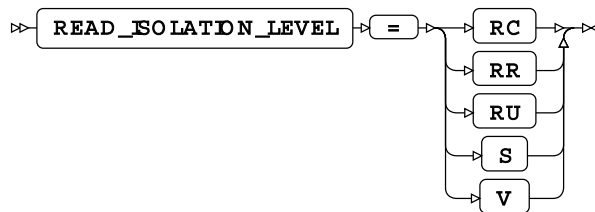
TRACEFILE



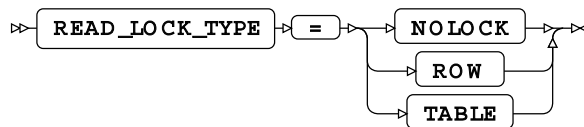
Type: String

Table locking options

READ_ISOLATION_LEVEL

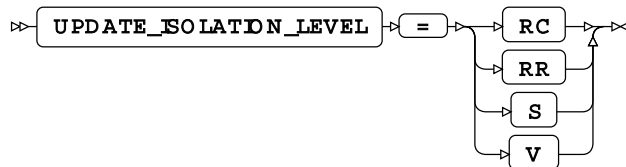


READ_LOCK_TYPE

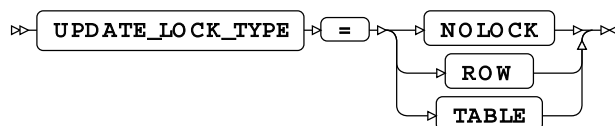


Default value: NOLOCK

UPDATE_ISOLATION_LEVEL



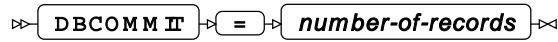
UPDATE_LOCK_TYPE



Default value: NOLOCK

ODBCOLD Dataset Options

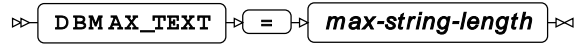
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



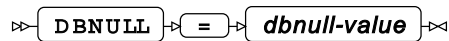
Type: Numeric

Minimum value: 1

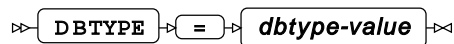
Maximum value: 32767

Default value: 4000

DBNULL



DBTYPE

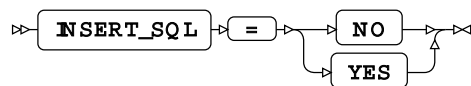


IGNORE_READ_ONLY_COLUMNS

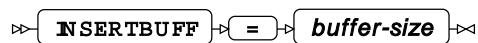


Default value: FALSE

INSERT_SQL



INSERTBUFF

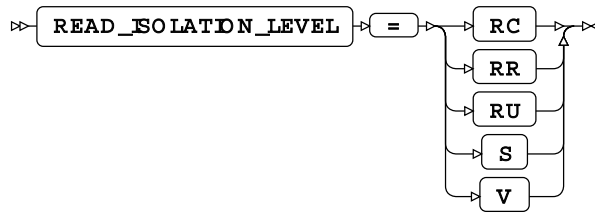


Type: Numeric

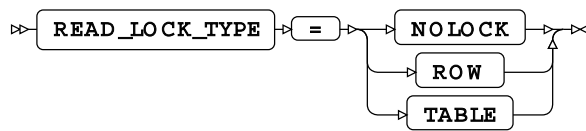
Minimum value: 1

Maximum value: 32767

READ_ISOLATION_LEVEL



READ_LOCK_TYPE



Default value: NOLOCK

READBUFF

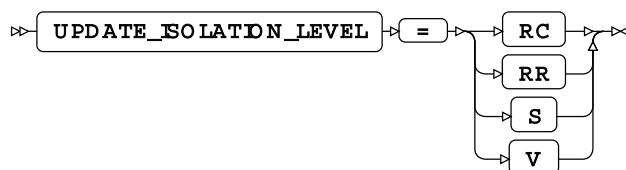


Type: Numeric

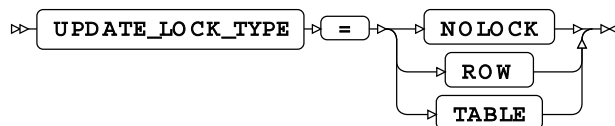
Minimum value: 1

Maximum value: 32767

UPDATE_ISOLATION_LEVEL



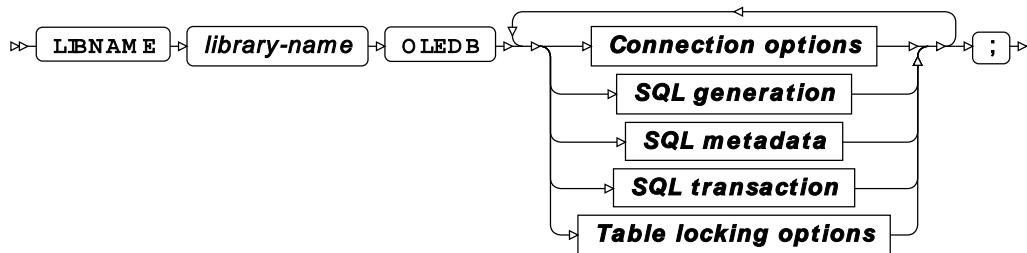
UPDATE_LOCK_TYPE



Default value: NOLOCK

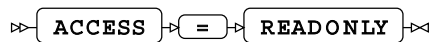
WPS Engine for OLEDB

OLEDB

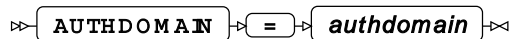


Connection options

ACCESS

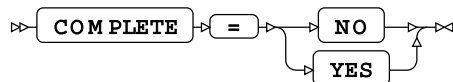


AUTHDOMAIN

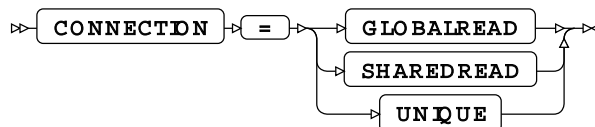


Type: String

COMPLETE

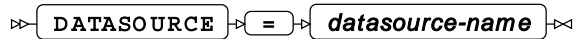


CONNECTION



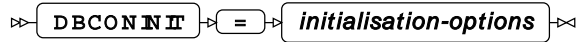
Default value: SHAREDREAD

DATASOURCE



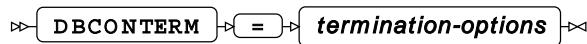
Type: String

DBCONINIT



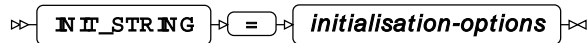
Type: String

DBCONTERM



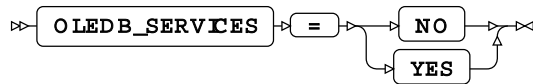
Type: String

INIT_STRING

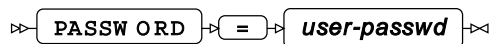


Type: String

OLEDB_SERVICES

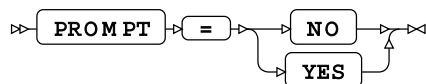


PASSWORD

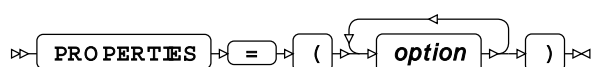


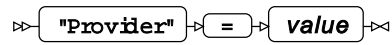
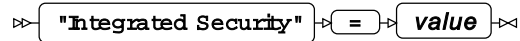
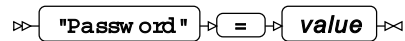
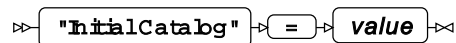
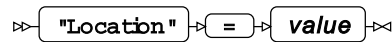
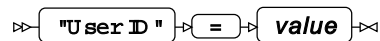
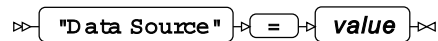
Type: String

PROMPT

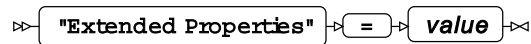


PROPERTIES



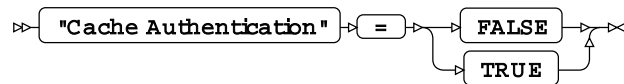
*option***"Provider"****Type:** String**"Integrated Security"****Type:** String**"Password"****Type:** String**"Initial Catalog"****Type:** String**"Location"****Type:** String**"User ID"****Type:** String**"Data Source"****Type:** String

"Extended Properties"

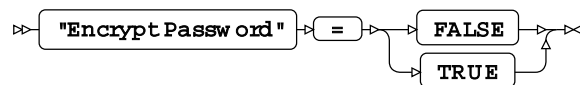


Type: String

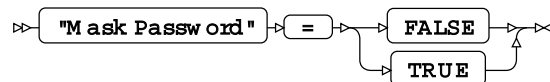
"Cache Authentication"



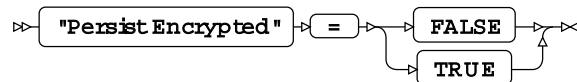
"Encrypt Password"



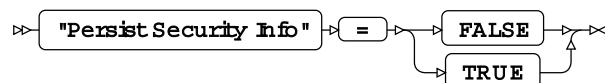
"Mask Password"



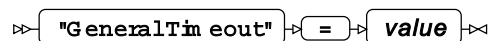
"Persist Encrypted"



"Persist Security Info"

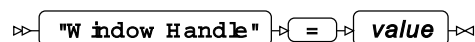


"General Timeout"

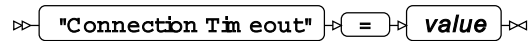


Type: Numeric

"Window Handle"



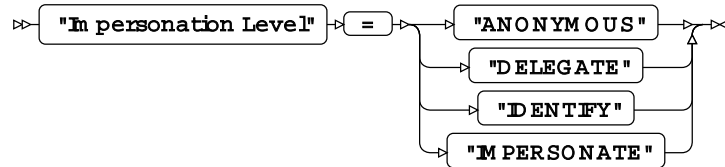
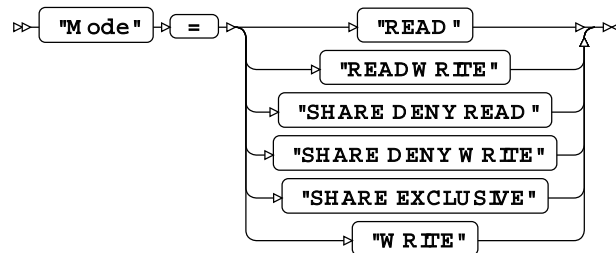
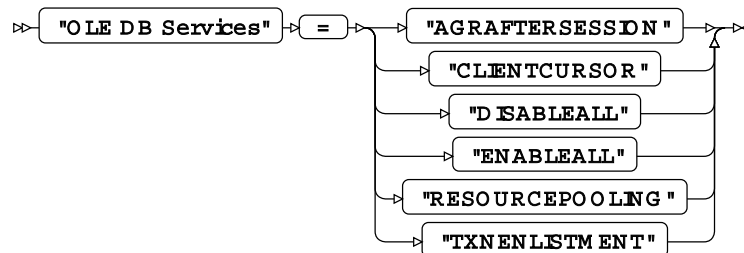
Type: Numeric

"Connection Timeout"

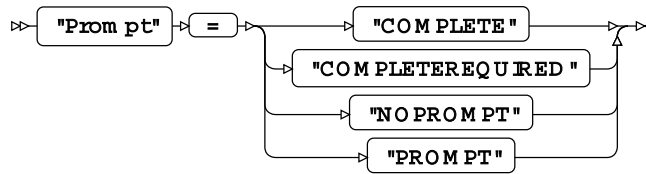
Type: Numeric

"Locale Identifier"

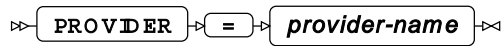
Type: Numeric

"Impersonation Level"**"Mode"****"OLE DB Services"**

"Prompt"

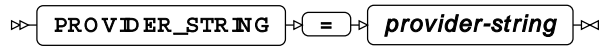


PROVIDER



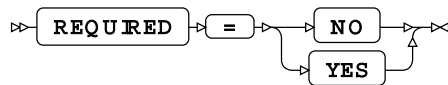
Type: String

PROVIDER_STRING

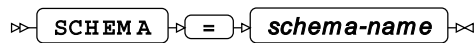


Type: String

REQUIRED

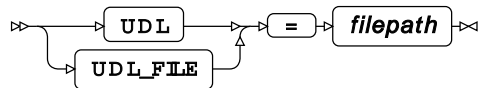


SCHEMA



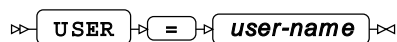
Type: String

UDL



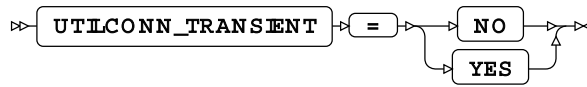
Type: String

USER



Type: String

UTILCONN_TRANSIENT



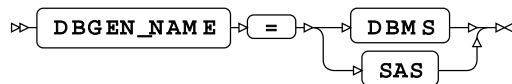
SQL generation

DBCREATE_TABLE_OPTS



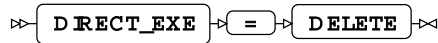
Type: String

DBGEN_NAME

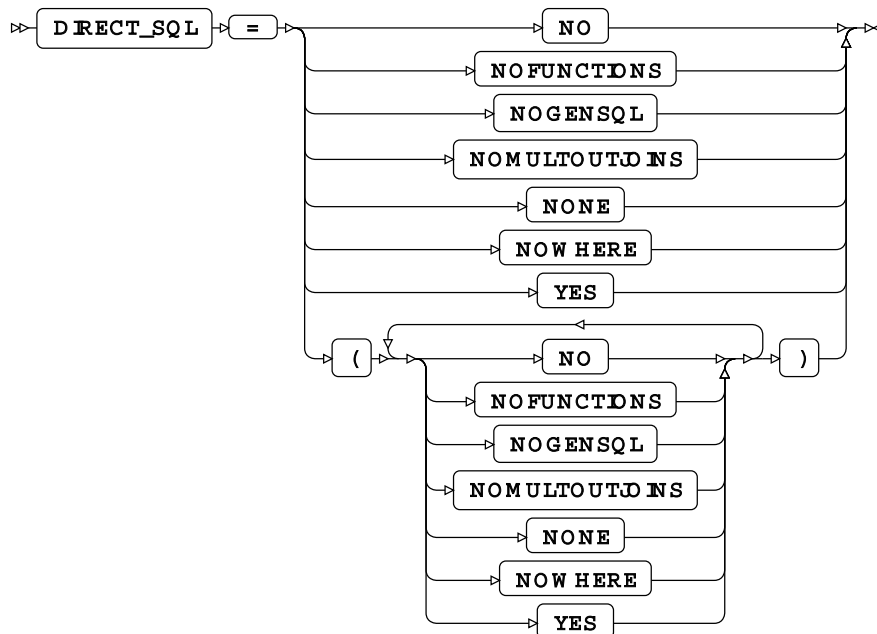


Default value: SAS

DIRECT_EXE

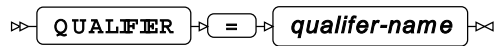


DIRECT_SQL



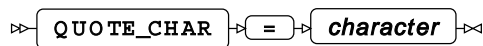
Default value: YES

QUALIFIER



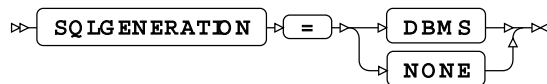
Type: String

QUOTE_CHAR

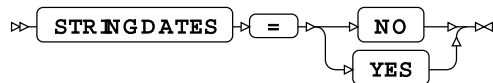


Type: String

SQLGENERATION

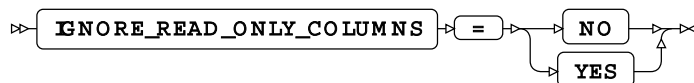


STRINGDATES



SQL metadata

IGNORE_READ_ONLY_COLUMNS



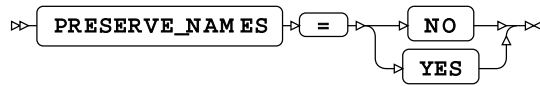
Default value: NO

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

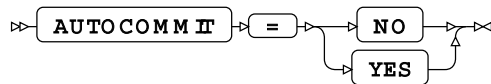
PRESERVE_TAB_NAMES



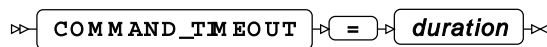
Default value: NO

SQL transaction

AUTOCOMMIT



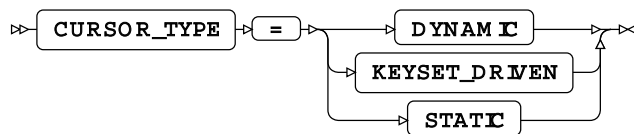
COMMAND_TIMEOUT



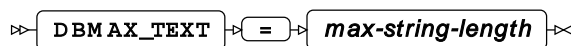
Type: Numeric

Minimum value: 0

CURSOR_TYPE



DBMAX_TEXT

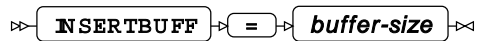


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF



Type: Numeric

Minimum value: 1

Maximum value: 32767

READBUFF



Type: Numeric

Minimum value: 1

Maximum value: 32767

SPOOL

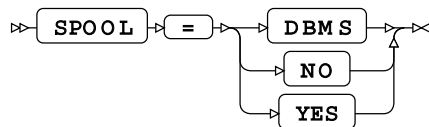
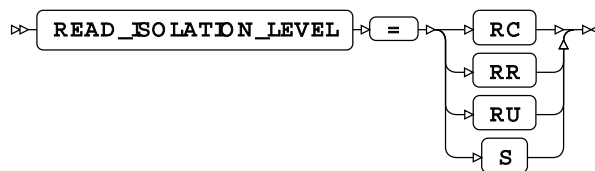
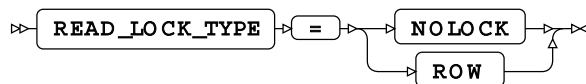


Table locking options

READ_ISOLATION_LEVEL



READ_LOCK_TYPE

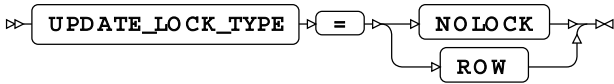


Default value: NOLOCK

UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE



Default value: NOLOCK

WPS Engine for Oracle

Two versions of the Oracle engine are available:

- A multi-threaded version. This is the latest version of the engine and is activated using the keyword ORACLE.
- A single threaded version. This was the original version of the engine, and is activated using the keyword ORACLEOLD.

Data Types in Oracle ↗	3649
Describes the correspondence between WPS formats and Oracle data types.	
How to use the Oracle engine ↗	3651
Connect to an Oracle database in a SAS language program.	
ORACLE connection reference ↗	3652
Describes the syntax and options for Oracle connection statements.	

Data Types in Oracle

Describes the correspondence between WPS formats and Oracle data types.

This section describes the correspondence between WPS formats and Oracle data types. WPS has many formats that affect the output and display of data. When you write data to an Oracle database table using the Oracle data engine, formatted data is converted to an equivalent and sensible data type. Many formats only affect the layout of data output, such as adding currency symbols or comma separators, and these formats have no effect when writing data to Oracle. See the sections below for more details.

Unformatted data

WPS format	Resulting Oracle data type	Notes
Unformatted number	NUMBER	
Unformatted string	VARCHAR2 (<i>x</i>)	<i>x</i> is a multiple of four. Each character in the string is allotted four bytes, which provides space for characters from a multibyte character set. For example, if the string in the dataset is one character long, the Oracle datatype is VARCHAR2 (4) ; if the string is two characters long, the datatype is VARCHAR2 (8) ; and so on.

Formatted data - numbers

WPS format	Resulting Oracle data type	Notes
<i>w. d</i>	NUMBER (<i>w</i> , <i>d</i>)	The width and number of decimal places are replicated in the Oracle data type. For example, data with the number format 4 . 2 is reproduced as the data type NUMBER (4, 2) .
BEST. and BEST <i>w.</i>	NUMBER	The number is passed to the database unformatted.
FLOAT <i>w. d</i>	NUMBER	The number is passed to the database unformatted.

WPS also provides NLS-sensitive money and numeric formats; the Oracle datatype of these corresponds to the equivalent basic numeric format above.

Formatted data - strings

WPS format	Oracle data type	Notes
\$ <i>w.</i> \$CHAR <i>w.</i> \$F <i>w.</i>	VARCHAR2 (<i>x</i>)	<i>x</i> is a multiple of four. Each character in the string is allotted four bytes, which provides space for characters from a multibyte character set. For example, if the string in the dataset is one character long, the Oracle datatype is VARCHAR2 (4) ; if the

WPS format	Oracle data type	Notes
		string is two characters long, the datatype is VARCHAR2 (4); and so on.

Formatted data - dates and times

WPS format	Oracle data type	Notes
DATE _w .	DATE	
DDMMYY _w . and all variants (such as DDMMYYB _w ., MMDDYY _S _w ., and YYMM _w .)	DATE	
DTDATE _w . and all variants (such as DTMONYY _w . and DTWKDATX _w .)	TIMESTAMP (6)	
TIME _w . , HOUR _w ., HHMM _w . and all similar time formats.	TIMESTAMP (6)	
JULIAN _w . and all similar date formats.	DATE	

WPS also provides international and NLS-sensitive date formats; the Oracle datatype of these corresponds to the equivalent basic format above.

How to use the Oracle engine

Connect to an Oracle database in a SAS language program.

In this example, a connection is made to the database using the Oracle multi-threaded engine:

```
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' USER=ARichards PASSWORD *****;
```

The Transparent Network Substrate name (TNS-name) is created on a computer when an Oracle server exists on a remote site. The TNS-name enables a connection between a client and server.

To connect to a database using a single user thread Oracle engine, substitute ORACLEOLD for ORACLE in the LIBNAME statement:

```
LIBNAME MyLib ORACLEOLD PATH='oracle-pz-e27/ZY' USER=ARichards PASSWORD *****;
```

ORACLE connection reference

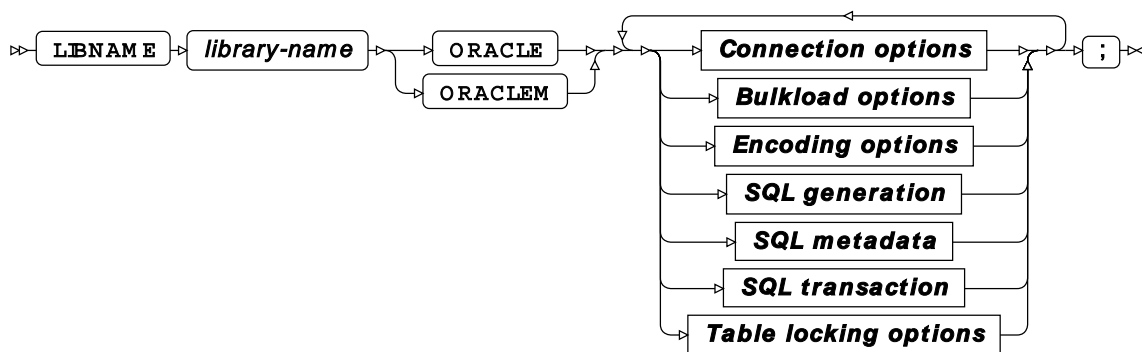
Describes the syntax and options for Oracle connection statements.

The `LIBNAME` library connection statement provides access to an Oracle database within a SAS language program. A full description is provided of how both the multi-threaded engine and the single-threaded engine access a database, and the options available to those engines. Options for both engines are similar, but where they differ an explanation is provided.

ORACLE ↗	3652
The <code>LIBNAME</code> library connection statement provides a connection to an Oracle database using the multi-threaded engine.	
ORACLE Dataset Options ↗	3680
ORACLEOLD ↗	3693
The <code>LIBNAME</code> library connection statement provides access to an Oracle database using the single-threaded engine.	
ORACLEOLD Dataset Options ↗	3718

ORACLE

The `LIBNAME` library connection statement provides a connection to an Oracle database using the multi-threaded engine.



The `LIBNAME` library reference enables a program to access to the database using the name defined in the `library-name` argument. You can use the specified library reference in any SAS language programs to access data stored in the database, as long as programs are run in the same WPS session as that in which the library reference was specified. The library reference is only active during the current WPS session. The `LIBNAME` statement contains options that, when specified, determine how SAS language programs interact with the database, grouped as follows:

- Connection options [↗](#) (page 3653): Connect to the Oracle database.
- Bulkload options [↗](#) (page 3658): Rapidly insert large amounts of data into a database using bulk loading (bulk insert).

- Encoding options [↗](#) (page 3665): Manage the encoding differences between database and WPS client.
- SQL generation options [↗](#) (page 3668): Affect how SQL statements are created, and whether the statements are processed by the database or WPS.
- SQL metadata options [↗](#) (page 3673): Determine how table description information or query statements are formatted and used.
- SQL transaction options [↗](#) (page 3675): Affect how SQL statements consisting of execution and data integrity are passed between the Oracle server and WPS.
- Table locking options [↗](#) (page 3677): Determine how WPS interacts with the Oracle table and row locking mechanisms.

You can also specify options for individual tables. These override the same options set on the `LIBNAME` statement. For more information, see *ORACLE dataset options* [↗](#) (page 3680).

library-name

The name used in other SAS language statements to access the database.

For example, the following statement:

```
LIBNAME myLib ORACLE PATH='TNS-name' USER=user-name PASSWORD=user-password;
```

creates a connection to a database using the name `MyLib`. This name can then be used in, for example, the `SQL` procedure:

```
PROC SQL;  
  INSERT INTO MyLib.person VALUES (32, 'Smith', 'John', 479216691);  
QUIT;
```

Connection options

These options provide the basic access settings for an Oracle database.

ACCESS

Specifies the access mode for the library connection.

➤ **ACCESS** ➤ **=** ➤ **READONLY** ➤

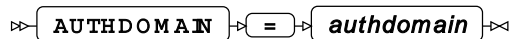
READONLY

The library connection can only be used to read data. Specifying `ACCESS = READONLY` overrides insert or update settings in other options and can result in data not being modified as expected.

If this option is not specified, the library connection uses a *read-write* access mode enabling read, insert, and update operations.

AUTHDOMAIN

Specifies the authorisation domain.



Type: String

The authorisation domain provides permissions to access a database server. WPS uses Hub as an authorisation domain, and a Hub server must be available to your system.

In this example, permissions for accessing the Hub are supplied as system options, and the name of the authorisation domain containing the authorisation details in the Hub is specified to AUTHDOMAIN.

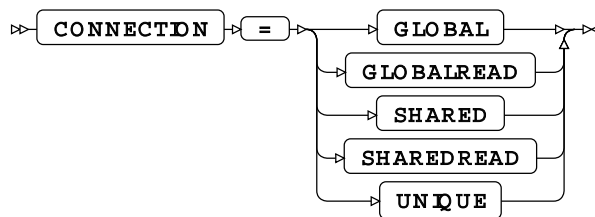
```
OPTIONS HUB_SERVER='blue_streak' HUB_PORT=309 HUB_PROTOCOL='HTTP'
HUB_USER='ARichards' HUB_PWD='*****';
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' AUTHDOMAIN='OracleAuth';
```

Note:

If USER and PASSWORD are specified in the LIBNAME statement, then AUTHDOMAIN is ignored.

CONNECTION

Specifies the type of connection to the database to use.



Default value: SHAREDREAD

Note:

If a database can only accept one SQL statement per connection, the connection type is always treated as a UNIQUE connection, whatever value is specified to this option.

GLOBAL

A single connection to the database is created for this library connection statement, and for any other libraries that match the connection options. This connection to the database is used for all read, insert, and update operations that use this library connection statement.

GLOBALREAD

A single connection to a database is created for this library connection statement, and for any other libraries that match the connection options. This connection to the database is used for all read operations. However, a separate connection to the database is created for each write operation.

SHARED

All database operations using this library connection statement share the same default connection created when the library connection statement is invoked. A software locking mechanism enables information to be updated. Shared connections are not currently supported.

SHAREDREAD

All database read operations that use this library connection statement share the same connection to the database that is created when the library connection statement is invoked. A connection to the database is created when the first insert or update operation occurs, and then this connection is shared for subsequent updates to the database.

This option interacts with the `UTILCONN_TRANSIENT` option as follows:

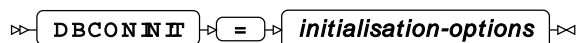
- If `UTILCONN_TRANSIENT = YES`, a utility connection is created each time the connection to the database is required, but not persisted for future connections.
- If a utility connection does not exist, and `UTILCONN_TRANSIENT = NO`, a utility connection is created and stored for future connections to the database.
- If a utility connection already exists, and `UTILCONN_TRANSIENT = NO` is specified, the existing utility connection is used and then stored for future connections to the database.

UNIQUE

Each operation using this library connection statement has its own connection to the database.

DBCONINIT

Specifies SQL statements or database commands that are processed every time the library connection is opened.



Type: String

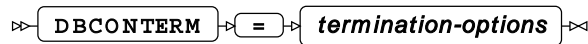
The connection is opened when this `LIBNAME` statement is first executed, and then every time a connection to the database is made; for example, by running an SQL statement in the SQL procedure.

In this example, a connection is created to an Oracle database. An existing table `UK_PMJan` is deleted, if it exists:

```
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' USER=test PASSWORD=*****  
DBCONINIT='DROP TABLE IF EXISTS UK_PMJan';
```

DBCINTERM

Specifies SQL statements or database commands that are processed every time the library connection is closed.



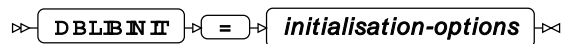
Type: String

In this example, a new table UK_PMFeb is created and the table UK_PMJan is deleted, if it exists.

```
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' USER=test  
PASSWORD=***** DBCONTERM='DROP TABLE IF EXISTS UK_PMJan';  
  
DATA myLib.UK_PMFeb;  
SET UK_PM;  
  
RUN;
```

DBLIBINIT

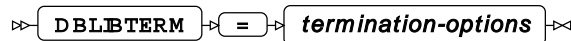
Specifies SQL statements or database commands that are processed after the first library connection has been successfully made.



Type: String

DBLIBTERM

Specifies SQL statements or database commands that are processed before the first library is disconnected.

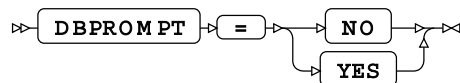


Type: String

DBPROMPT

For compatibility only.

This option currently has no effect.



Default value: NO

NO

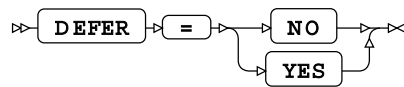
For compatibility only.

YES

For compatibility only.

DEFER

Specifies at what point a library connection is made.



A connection to the database can be made as soon as the library connection statement is executed, or only when an action is performed on the database.

NO

Connect when the library connection statement is executed.

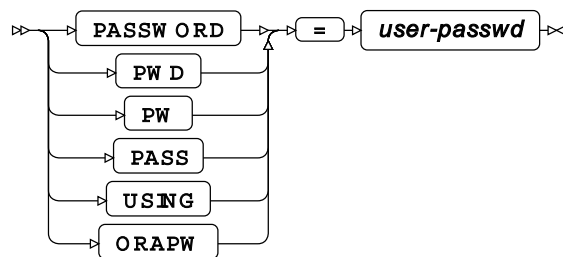
YES

Connect when a SAS language statement requires access to the database.

This is the default if `CONNECTION` [↗](#) (page 3654) is set to `UNIQUE`.

PASSWORD

Specifies the password for the user name.

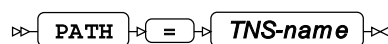


Type: String

If special characters are used as part of the string, you must enter *user-passwd* in quotation marks. The user name is specified with the `USER` option.

PATH

Specifies the TNS service name for the Oracle database.

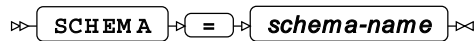


Type: String

The TNS name is an alias for the Oracle Call Interface (OCI) connection string that identifies the database server and instance to which you want to connect. For example: `PATH='oracle-pg-e27/ZY'`.

SCHEMA

Specifies the name of the database schema with which the connection interacts.

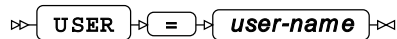


Type: String

The schema is a grouping of database objects and data accessible by the user that can be manipulated through SQL statements.

USER

Specifies the user name required to access the database.

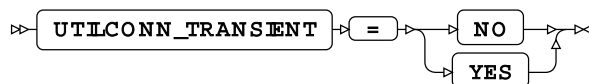


Type: String

If special characters are used as part of the string, you must enter *user-name* in quotation marks.

UTILCONN_TRANSIENT

Specifies whether the *utility connection* is transient or non-transient.



Default value: NO

A utility connection is created to separate data read and update processes from metadata handling. This connection is used by WPS Workbench to list the members of the database in which to populate information; for example, the *Sashelp* tables.

NO

The utility connection is created and stored for future connections to the database.

YES

The utility connection is created whenever required, but is not persisted for future connections to the database.

Bulkload options

Rapidly insert large amounts of data into a database using bulk loading (bulk insert). Bulk insert uses either the native Oracle Call Interface (OCI) or, if installed, SQL*Loader.

SQL*Loader provides a bulk insert mechanism that can be faster than the OCI, because indexes are only updated when data insertion has finished.

Note:

With SQL*Loader you can inadvertently insert incorrect data into an index, and create a table that cannot be accessed. You should therefore keep the intermediate data file that enables you to validate information on insertion.

Oracle supports two methods for bulk loading data:

- Conventional path load. SQL `INSERT` statements are constructed that encapsulate the data to be inserted. This is the default method used by both OCI or SQL*Loader.
- Direct path load. Data is bulk loaded directly using the OCI direct path API.

Library connection options enable you to specify which bulk loading methods to use. To bulk load data using:

- The conventional path and the OCI, specify `BULKLOAD=YES`.
- The conventional path and SQL*Loader, specify:

```
BULKLOAD=YES BL_USE_SQLLDR=YES
```

- The direct path and the OCI, specify:

```
BULKLOAD=YES BL_DIRECT_PATH=YES
```

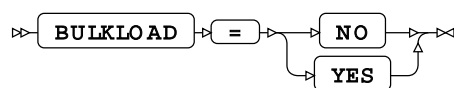
- The direct path and SQL*Loader, specify:

```
BULKLOAD=YES BL_USE_SQLLDR=YES BL_DIRECT_PATH=YES
```

For information on these options, see [BULKLOAD](#) (page 3659), [BL_DIRECT_PATH](#) (page 3661) and [BL_USE_SQLLDR](#) (page 3665).

BULKLOAD

Specifies whether the library connection permits bulk insert into a database table.



NO

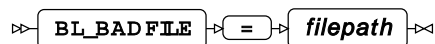
Data cannot be bulk inserted. All other bulkload options are ignored.

YES

Data can be bulk inserted.

BL_BADFILE

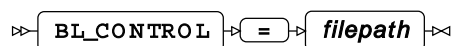
Specifies the path to the SQL*Loader bad file.



The bad file is used to store records that could not be inserted into the database table due to errors. The file is only created if required. The file has the extension or type `*.bad`.

BL_CONTROL

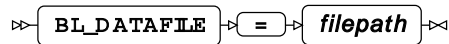
Specifies the path to the SQL*Loader control file.



The control file is used to specify how data is loaded from the data file specified in `BL_DATAFILE` into the database table. The file has the extension or type `*.ctrl`.

BL_DATAFILE

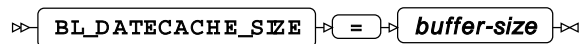
Specifies the path to the SQL*Loader data file.



The data file contains the data to be inserted into the database table. The file has the extension or type `*.dat`.

BL_DATECACHE_SIZE

Specifies the size of the date cache to use with a bulk insert using OCI.



This cache is used when data is inserted that contains date information. A date value needs to be converted from a SAS language date format to an Oracle data type before it is inserted into a table. A converted date value is stored in this cache to speed insertion of duplicate information.

BL_DEFAULT_DIR

Specifies the default path to use for the files automatically generated by SQL*Loader (bad, data, discard, log, and parameter).

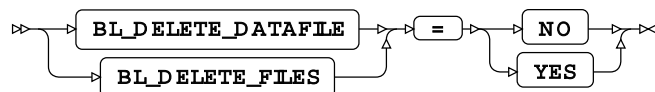


Type: String

If pathnames for these automatically generated files are not specified using the corresponding bulkload options, a file is automatically created when required and stored in the location specified by this option.

BL_DELETE_DATAFILE

Specifies whether the SQL*Loader data and associated control and log files are deleted.



Default value: YES

NO

Keep the data, control, and log files after insert.

YES

Delete the data, control, and log files after insert.

BL_DELETE_ONLY_DATAFILE

Specifies whether only the SQL*Loader data file is deleted.



Default value: NO

NO

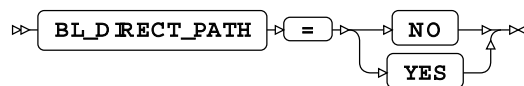
Keep the data after insert. The control and log files are also kept.

YES

Delete only the data file after insert.

BL_DIRECT_PATH

Specifies that the direct path load is used, rather than a conventional path load, to bulk insert data.



Default value: NO

NO

Use a conventional path load.

YES

Use a direct path load. This method can be used with either the Oracle Call Interface or SQL*Loader. To use it with SQL*Loader, also specify `BL_USE_SQLLDR=YES`.

BL_DISCARDFILE

Specifies the path to the SQL*Loader discard file.



The discard file contains data that has neither been inserted into the database table, nor been rejected as bad after the bulk insert process. The file has the extension or type `*.dsc`.

BL_INDEX_OPTIONS

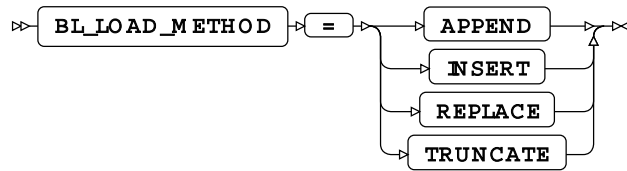
Specifies how indexes are created when inserting information through SQL*Loader.



Type: String

BL_LOAD_METHOD

Specifies the method used by SQL*Loader to bulk insert data into a database table.



APPEND

Add data to an empty table or append data to a table that already contains rows.

INSERT

Add data to an empty table. Attempting to insert data into a non-empty table results in an error.

REPLACE

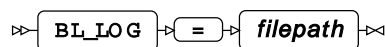
Replace all existing content in the database table with the content of the SQL*Loader data file. An SQL `DELETE FROM TABLE...` statement is first executed, and any delete triggers created on the table are therefore run before the new data is inserted.

TRUNCATE

Replace all existing content in the database table with the content of the SQL*Loader data file. A `TRUNCATE TABLE...` statement is executed as the first step to reset the number of table rows to zero. Any referential constraints on the table must be disabled before using this load method.

BL_LOG

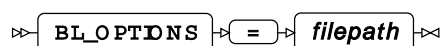
Specifies the path to the SQL*Loader log file.



The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type `*.log`.

BL_OPTIONS

Specifies options for SQL*Loader.

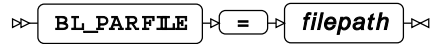


Type: String

Where these options are specified in both the SQL*Loader control file and `BL_OPTIONS`, any options specified using `BL_OPTIONS` take priority.

BL_PARFILE

Specifies the path to the SQL*Loader parameter file.



This file is used to store frequently-used command line options. The file has the extension or type `*.par`.

BL_PRESERVE_BLANKS

Specifies whether SQL*Loader trims trailing spaces from inserted data.



Default value: NO

NO

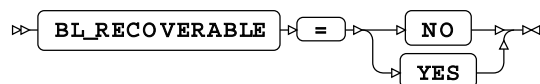
Remove trailing spaces.

YES

Do not remove trailing spaces.

BL_RECOVERABLE

Specifies whether the SQL*Loader bulk insert attempts to reload data if there is a problem during the insert that is not related to data.



Default value: YES

NO

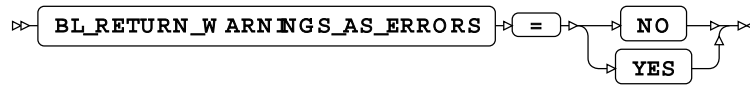
Do not attempt to recover from errors. No redo log is created.

YES

Attempt to recover from an error. The data is added to the redo log, and this log is used as the basis of the recovery.

BL_RETURN_WARNINGS_AS_ERRORS

Specifies whether warnings generated when bulk-inserting data with SQL*Loader are displayed as errors in the SQL*Loader log file.



Default value: NO

NO

Do not display warnings as errors in the SQL*Loader log file.

YES

Display warnings as errors in the SQL*Loader log file.

BL_SQLLDR_PATH

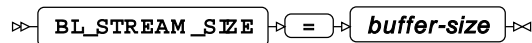
Specifies the path of the SQL*Loader application.



Type: String

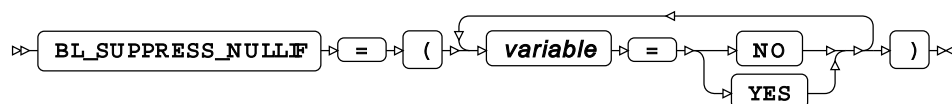
BL_STREAM_SIZE

Specifies the size of the buffer to use when bulk loading data using the SQL*Loader and direct path loading.



BL_SUPPRESS_NULLIF

Specifies whether to ignore NULLIF conditions defined in the SQL*Loader control file.



Default value: NO

variable

The name of a column (variable), or `_ALL_`.

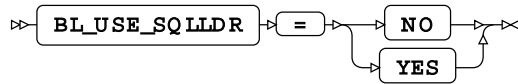
Specify:

- `variable=YES` to ignore the NULLIF condition for the column `variable`
- `variable=NO` to ignore the NULLIF condition for the column `variable`; this is the default.
- `_ALL_=YES` to ignore NULLIF conditions for all columns,

If you specify `_ALL_=YES`, do not specify subsequent arguments. If you do, the list of arguments might not execute correctly.

BL_USE_SQLLDR

Specifies whether bulk inserts are made using Oracle's SQL*Loader.



Default value: NO

Note:

SQL*Loader is not included in the Oracle Instant Client and must be installed separately.

NO

Do not use SQL*Loader for bulk inserts; the Oracle Call Interface (OCI) is used instead.

YES

Use SQL*Loader for bulk inserts.

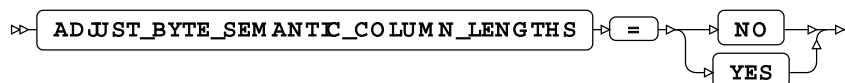
The option `BL_DIRECT_PATH` [↗](#) (page 3661) can be used to specify whether SQL*Loader uses the conventional load path or the direct load path. See *Bulkload options* [↗](#) (page 3658)

Encoding options

Options that manage the encoding differences between database and WPS client.

ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS

Specifies how to handle database columns defined as `CHAR` and `VARCHAR` when WPS and database server encodings differ.



If the encodings between the database server and WPS differ, and database columns have been specified as the `CHAR` and `VARCHAR` client column size is adjusted to store all characters retrieved from the database server.

The default depends on the value of `DBCLIENT_MAX_BYTES`. If `DBCLIENT_MAX_BYTES` is equal to 1, then the default is NO. If `DBCLIENT_MAX_BYTES` is greater than 1, then the default is YES.

NO

The WPS variable holds the same number of bytes as the corresponding column in the database. If the number of bytes used to represent a character differs between the character sets used by WPS and by the database, data might be truncated.

YES

Adjust variable lengths in WPS to ensure all characters from the corresponding column are accurately represented.

For example, if the character set on the database server uses two bytes to represent a character, a 500 character column is returned as 1000 bytes. If the character set in WPS uses four bytes to represent a character, the client reads the number of characters (1000/2 = 500 characters) and allocates the correct amount of space (500*4=2000 bytes) to ensure the characters are correctly encoded.

ADJUST_NCHAR_COLUMN_LENGTHS

Specifies how to handle database columns defined as `NCHAR` and `NVARCHAR`.



Default value: YES

NO

The WPS variable holds the same number of bytes as the corresponding column in the database. If the number of bytes used to represent a character differs between the character sets used by WPS and by the database, data might be truncated.

YES

The column length in WPS is adjusted to cater for the value defined in `DBSERVER_MAX_BYTES`. The adjustment is calculated using:

number_of_characters * *value_of_DBCLIENT_MAX_BYTES*

where *number_of_characters* is the number of characters in the column, and *value_of_DBCLIENT_MAX_BYTES*

where *number_of_characters* is the number of characters in the column, and *value_of_DBCLIENT_MAX_BYTES* is the value specified by `DBCLIENT_MAX_BYTES` [↗](#) (page 3667).

DB_LENGTH_SEMANTICS_BYTE

Specifies how to handle database columns defined as `NCHAR` and `NVARCHAR`.



Default value: TRUE

NO

Definitions for `CHAR` or `VARCHAR2` columns are created by specifying the total number of characters.

For example, if a dataset has a column that has 20 bytes available to store characters, and each character is composed of:

- One byte, then 20 characters can be stored in each column.
- Two bytes, then only 10 characters can be stored in each column.
- Four bytes, then only five characters can be stored in each column.

The length defined for `CHAR` or `VARCHAR2` allows a maximum number of characters to be stored whatever the character set encoding used on the database. For example, if the character set consists of a fixed two-byte encoding, and the variable is 20 bytes long, then the data type for the variable is the length of the variable, divided by two bytes per character. The data type for the corresponding column is therefore `VARCHAR2 (10 CHAR)`. If a character set uses a variable number of bytes to define a character, then the maximum number of bytes used to define a character is used to calculate the data type; for example, characters in the UTF-8 character set can be represented using up to four bytes, therefore a 20 byte variable is defined as `VARCHAR2 (5 CHAR)` in the database.

YES

Definitions for `CHAR` or `VARCHAR2` data types are created by specifying the data type in bytes. The length of the data type is dependent on the encoding used by the database server, and whether characters have fixed or variable encodings. It is calculated as:

number_of_characters * *value_of_DBSERVER_MAX_BYTES*

where *number_of_characters* is the number of characters defined for the variable, and *value_of_DBSERVER_MAX_BYTES* is the value specified by `DBSERVER_MAX_BYTES` [\(page 3668\)](#).

For example, assume a dataset has a column of 20 bytes available to store characters. The dataset variable is defined as 10 characters long, and the database character set has four bytes UTF-8 encoding, as specified in `DBSERVER_MAX_BYTES`). The equivalent table column data type is defined as `VARCHAR (40)` (that is, 10 x 4).

DBCLIENT_MAX_BYTES

Specifies the maximum number of bytes used to encode each character in WPS.

➤ `DBCLIENT_MAX_BYTES` ➤ = ➤ `bytes-per-character` ➤

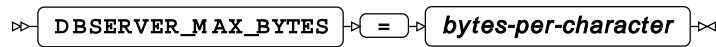
Type: Numeric

Minimum value: 1

When not set, this defaults to the number of bytes used by the current WPS session encoding.

DBSERVER_MAX_BYTES

Specifies the maximum number of bytes per character in the encoding used by the database.



Type: Numeric

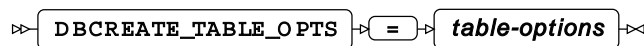
Minimum value: 1

SQL generation

Options that affect how SQL statements are created, and whether the statements are created on the Oracle server or by WPS.

DBCREATE_TABLE_OPTS

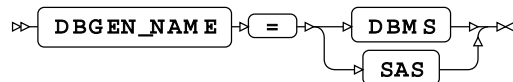
Specifies extra table options to be added to a `CREATE TABLE...` statement after the columns in the table have been defined.



Type: String

DBGEN_NAME

Specifies how to modify column names that require adjustment to be valid in the language of SAS.



Default value: SAS

DBMS

If the column name contains invalid characters, those characters are replaced with an underscore. If this change results in a name that clashes with that of another column, the column count is appended to the column name.

For example, if your table contains the columns `id$x`, `id#x` and `id_x`, the variables in the language of SAS would be `ID_X`, `ID_X0`, and `ID_X1` respectively.

Note:

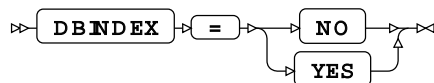
The numbering starts at the first repeat of the variable name, and also begins at 0.

SAS

If the column name contains invalid characters, or have the same name, the column name is replaced with the string `_COL`. If this change results in a name clash with other columns, the column count is appended to the name: `_COL x` where x is the column, starting at 0 (zero).

DBINDEX

Specifies whether database indexes are used, where available, to provide a fast table look-up when joining a table with a WPS dataset. Database indexes are only used in this way when an available index has the same fields as the join keys.



Default value: NO

NO

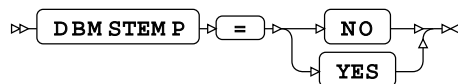
Do not use database indexes.

YES

Use database indexes.

DBMSTEMP

Specifies whether to create temporary database tables rather than permanent tables.



Default value: NO

Temporary tables are deleted on disconnection.

NO

Create temporary database tables.

YES

Do not create temporary database tables.

A typical `DATA` step might be:

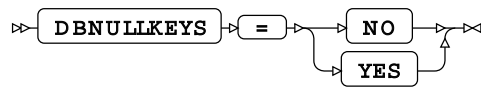
```
LIBNAME tdtemp ORACLE path='oracle-xe-p21/XZ' user=test
password=c987dZ4y connection=global dbmstemp=YES;

data carinfo.customer_update;
set tdtemp.customer_update;

RUN;
```

DBNULLKEYS

Specifies what should happen if a value in a column specified as a key using `DBKEY` is null.



`DBKEY` is a dataset option.

NO

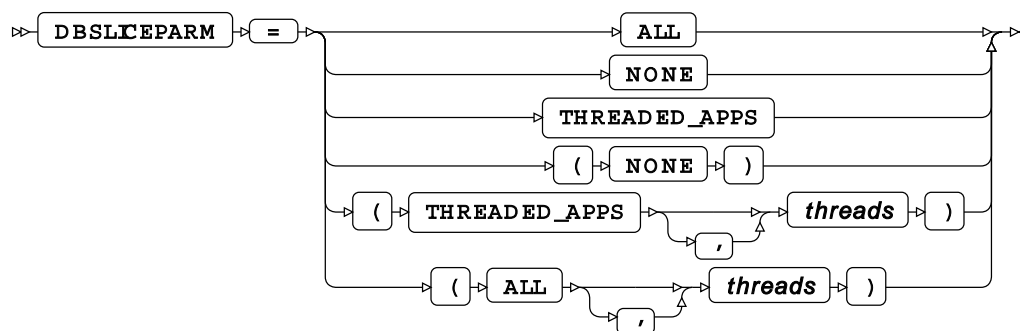
Specifies that if the value in a column defined as a key column using `DBKEY` is null, the row that contains the null value is ignored.

YES

Specifies that if the value in a column defined as a key column using `DBKEY` is null, the row that contains the null value is processed.

DBSLICEPARM

Specifies whether to perform multi-threaded reads, in which a part of the database table is returned on each thread.



This option can be used to improve the performance of read operations on large database tables. You can specify a number of threads to be used; different parts of the table are then returned on each thread.

Note:

The default value is that set by the system option `DBSLICEPARM`. The default if the system option has not been set is `THREADED APPS, 2`.

ALL

The number of available threads is equal to the number of available CPUs. To limit the number of threads used, specify the number in *threads*.

NONE

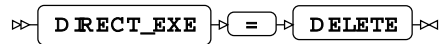
Multiple threads are not used. Data is returned on one thread.

THREADED_APPS

The number of available threads is equal to the number of available CPUs. To limit the number of threads used, specify the number in *threads*.

DIRECT_EXE

Specifies whether delete statements are executed directly on the database, or through WPS.



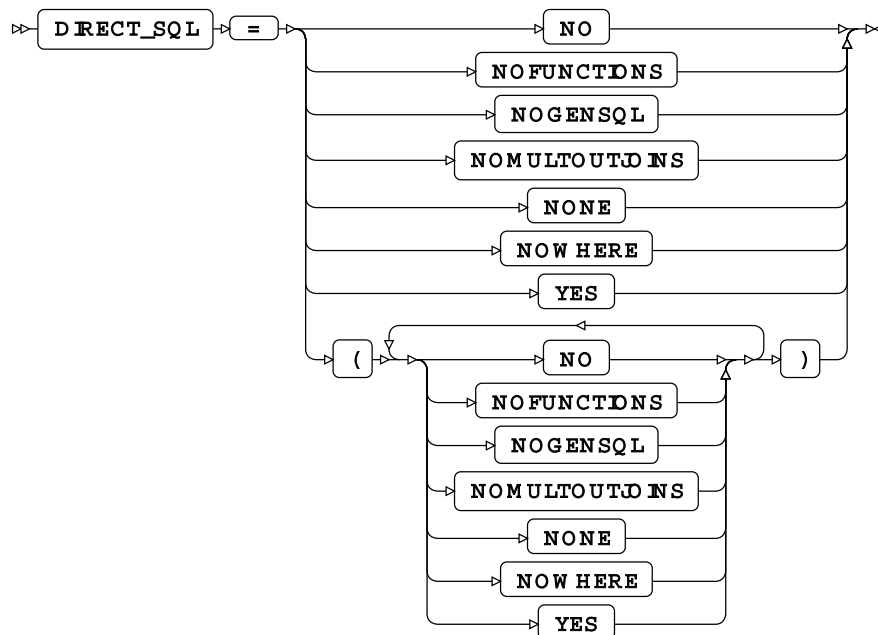
When not specified, the database table is scanned and rows matching the delete criteria are returned to WPS. An SQL `DELETE FROM...` statement is created by WPS for each affected row and passed to the database for processing.

DELETE

The SQL `DELETE FROM...` command is passed to the database and processed entirely by the database engine. In this case, the SQL `DELETE...` statements are not created by WPS.

DIRECT_SQL

Specifies whether SQL statements are passed through for processing by the database server, or processed by WPS.



Default value: YES

NO

All SQL statements are processed by WPS.

NOFUNCTIONS

Any SQL statements containing function calls are processed by WPS. All other statements that can be processed by the database server are passed through to the server for processing.

NOGENSQL

All SQL statements are processed by WPS.

NOMULTOUTJOINS

Any SQL statements containing multiple outer joins are processed by WPS. All other statements that can be processed by the database server are passed through to the server for processing.

NONE

All SQL statements are processed by WPS.

NOWHERE

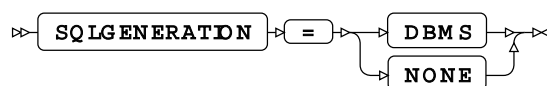
Any statements containing `WHERE` clauses are processed by WPS. All other statements that can be processed by the database server are passed through to the database server for processing.

YES

All SQL statements that can be processed in-database are passed through to the database server for processing.

SQLGENERATION

Specifies where the SQL statements required to create summary data for the SUMMARY procedure are processed.



This option enables you to specify whether the SQL statements required to create summary data for the SUMMARY procedure are generated and the data processed by the database server or by WPS.

DBMS

Data processing is carried out by the database server.

NONE

WPS generates the SQL statements and processes data, sending the result to the database server.

SQL metadata

Options used to determine how table description information or query statements are formatted and used.

PRESERVE_COL_NAMES

Specifies whether the case of variable names in datasets is preserved in column names in tables.



Default value: NO

Note:

When column names are enclosed in quotes, the database preserves the case.

NO

Case is not preserved. The name reverts to the default case the database.

For example, if a dataset contains a variable labelled `xX`, and the dataset is written to a database table, the column is labelled `xx`.

YES

Case is preserved.

Note:

This option does not function with a case-insensitive database.

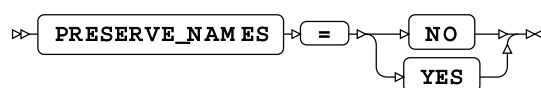
For example, the following program reads data from a table, and writes it to a dataset:

```
DATA MyLib.table1 (PRESERVE_COL_NAMES = YES);  
    SET Mynums;  
    PUT XX;  
RUN;
```

The dataset `Mynums` contains variables named `xX` and `yY`. The resulting table `table1` in the database specified by `MyLib` contains columns with names in the same case. In the `PUT` statement, however, the variable is specified in upper case; because the SAS language ignores the case of variables in statements, the value of the variable `xX` is written to the log.

PRESERVE_NAMES

Specifies whether both the table and column names in SQL statements are treated as case-sensitive or case-insensitive identifiers.



Default value: NO

NO

Table and column names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example if you have a table labelled `Customer`, this is typically treated as if it is labelled `CUSTOMER`. Any use of the table name, however it is written – `customer` or `Customer` – refers to the same table. The same rule applies to columns in the table; this means the same table and column combination is referenced in a SAS language program whichever case is used in the names.

YES

Table and column names in SQL statements are treated as case sensitive. As part of the statement processing, the names have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test...` becomes `SELECT "test"."myColumn" FROM "test"...`

Using case-sensitive names means that a combination of table and column labels within a SAS language program must accurately reflect the case of the names used within the database. For example, a statement such as `SELECT Test.Column1 FROM Test WHERE TEST.Column2...` would be interpreted as `SELECT "Test"."Column1" FROM "Test" WHERE "TEST"."Column2"...`, meaning the `WHERE...` clause in the in the statement has applied a different table than that in the `SELECT...` clause.

PRESERVE_TAB_NAMES

Specifies whether the table names in SQL statements are treated as case-sensitive – and converted to the normal label form for the database – or case-insensitive identifiers.



Default value: NO

NO

Table names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example, if you have a table labelled `myTable`, this is treated as if it is labelled `MYTABLE`. Any use of the table name in a SAS language program, however it is written – `MyTable` or `mytable` – refers to the same table.

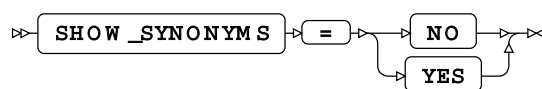
YES

Table names in SQL statements are treated as case sensitive. When a statement is processed, names are have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test` is interpreted as `SELECT "test".MYCOLUMN FROM "test"`.

Using case-sensitive names enables a database to contain multiple, similarly-named tables, because `clientName` and `CLIENTNAME` are treated as different tables.

SHOW_SYNONYMS

Specifies whether the Oracle-specific alternative name for database objects is used when querying the database structure.



Default value: NO

NO

The original object names are used when viewing the information about a database through `PROC DATASETS`, or when querying `DICTIONARY` tables through `PROC SQL`.

YES

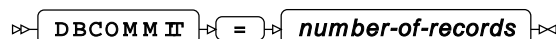
Created synonyms are used when viewing the information about a database through `PROC DATASETS`, or when querying `DICTIONARY` tables through `PROC SQL`.

SQL transaction

Options that affect how SQL statements consisting of execution and data integrity are passed between the Oracle server and WPS.

DBCOMMIT

Specifies the maximum number of records in each commit that is passed to the database server.

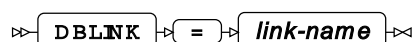


Type: Numeric

Minimum value: 0

DBLINK

Specifies a database link through which you can access objects on another database.



Type: String

link-name is a pointer to a table or view from a main Oracle database to a different Oracle database that might be on the same, or on a different, database server.

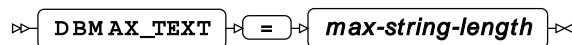
When this option is specified for a library connection, all SQL `SELECT...`, `INSERT...`, `UPDATE...`, and `DELETE...` statements that accessing the main database have the `DBLINK` name appended to the table or view name.

To access tables from both the main and linked databases requires two library connections to the same main database, and both connections must use the same connection credentials. One of the statements must have the `DBLINK` option specified and both statements require the `CONNECTION=GLOBAL` option specified.

The `DBLINK` reference is created using the OracleSQL statement `CREATE DATABASE LINK`.

DBMAX_TEXT

The maximum string length (number of characters) that can be inserted into a field in a database table.



Type: Numeric

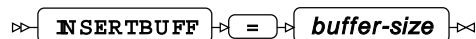
Minimum value: 1

Maximum value: 32767

Default value: 1024

INSERTBUFF

Specifies the number of records inserted into a database in a single insert action, when not using bulkload options. If `DBCOMMIT` is set, the lower value of `INSERTBUFF` or `DBCOMMIT` is used.



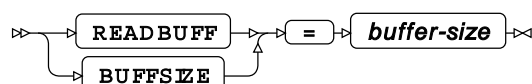
Type: Numeric

Minimum value: 1

Default value: 10

READBUFF

Specifies the number of records retrieved from the database in a single read action.



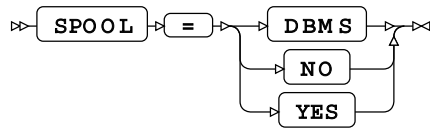
Type: Numeric

Minimum value: 1

Default value: 250

SPOOL

Spooling creates a temporary file containing the results of reading a database table.



This cache file is used for subsequent read actions on the data, enabling data to be read more quickly than re-reading the content from the database table.

DBMS

Spooling is active and handled by the DBMS.

NO

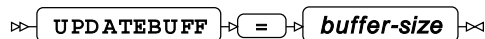
Spooling is active and handled by the DBMS.

YES

Spooling is active in WPS.

UPDATEBUFF

Specifies the number of records updated in or deleted from a database in a single action, when not using bulkload options. If `DBCOMMIT` is set, the lower value of `UPDATEBUFF` or `DBCOMMIT` is used.



Type: Numeric

Minimum value: 1

Default value: 1

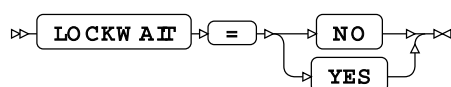
The maximum value is determined by the size of the database.

Table locking options

Options that determine how WPS interacts with the Oracle table and row locking mechanisms.

LOCKWAIT

Specifies what happens when another process attempts to read or update a locked database table.



Default value: YES

NO

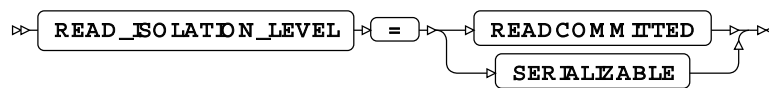
Nothing is read from or written to the table, and a message is written to the log.

YES

Wait for the lock to be removed from the table, and then attempt to read or update the table.

READ_ISOLATION_LEVEL

Specifies when a client can read data that is inserted into a database table if the table is accessed by multiple clients.



READ_ISOLATION_LEVEL settings only affect SQL SELECT statements, and then only when the ACCESS option is not set to READONLY.

READCOMMITTED

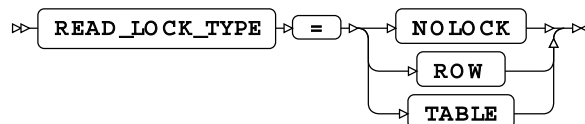
Only committed data is read by a client. Data inserted as part of an another client's transaction can be queried once the transaction is complete and the data has been committed to the database. Where multiple read actions take place against a table by the same client, returned results are only updated to reflect any transactional changes made between the read actions.

SERIALIZABLE

Where multiple clients are reading data from the table, only one query is processed at a time – any query of the table data must complete before the next query or data update can begin. If the client attempts to read data from the table while an update transaction is being processed, the client reading the table data cannot access the table until the updating transaction is complete.

READ_LOCK_TYPE

Specifies the type of lock to apply to read actions on a table.



Default value: NOLOCK

NOLOCK

No lock is applied to the table. There are no restrictions on updates to the table, and multiple, concurrent read actions of the data might return different results.

ROW

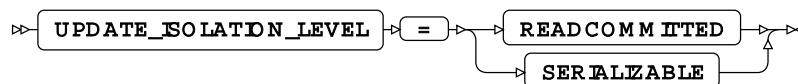
When `LOCKWAIT=NO`, the WPS client obtains a *row share* lock on a row. This lock allows concurrent access, but prevents a full table lock. This lock also prevents updates to the row until the read action is complete.

TABLE

When `LOCKWAIT=NO`, WPS obtains a *share* lock on the table. This lock allows concurrent read access, but prevents any updates to the table until the query action is complete.

UPDATE_ISOLATION_LEVEL

Specifies when a client can update a table if the table is accessed by multiple clients.



`UPDATE_ISOLATION_LEVEL` settings only affect SQL `UPDATE` statements, and then only when the `ACCESS` option is not set to `READONLY`.

READCOMMITTED

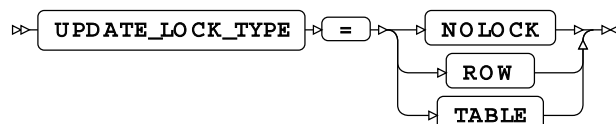
Read before update. A second read might return different data to the original read, as there is no restriction on the update which might insert new rows, delete existing rows, or change data.

SERIALIZABLE

A WPS client waits for the current client to finish its updates to a table before it makes its changes.

UPDATE_LOCK_TYPE

Specifies the type of lock to apply to update actions on a table in the database.



Default value: `NOLOCK`

NOLOCK

No lock is applied to the table. There are no restrictions on updates to the table, and multiple, concurrent read actions of the data might return different results.

ROW

The WPS client obtains a *row share* lock on the table and makes the required updates. This lock allows multiple client access to the table for reading and updating. None of those clients can obtain an *exclusive* lock on the table, and no other client can update the locked row until the transaction is complete.

TABLE

The WPS client obtains an *exclusive* lock on the table and makes the required updates. Any other activity on the table is delayed until updates are complete.

ORACLE Dataset Options

When dataset options are specified for a database table, the values of those options override the settings specified for the same options on the library connection statement. You should therefore configure a library connection statement so that it provides the best access to all accessible objects in a database. If different options are required for specific tables, the following dataset options can be used.

You can specify dataset options wherever you can specify a database table in a SAS language or SQL statement.

For example, dataset options can be added when a database table is specified in an `SQL INSERT` statement when using the SQL procedure:

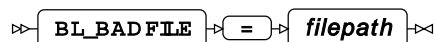
```
PROC SQL;
  INSERT INTO myLib.myTable (BULKLOAD=YES, BL_LOAD_METHOD=TRUNCATE)
    AS SELECT * FROM datasrc.append_src;
QUIT;
```

Similarly, dataset options can be added when a database table is specified as the input dataset in a SAS language program:

```
DATA myoutfile;
  SET mylib.a_xyz1 (BULKLOAD=YES);
RUN;
```

BL_BADFILE

Specifies the path to the SQL*Loader bad file.

 `BL_BADFILE` `=` `filepath`

The bad file is used to store records that could not be inserted into the database table due to errors. The file is only created if required. The file has the extension or type `*.bad`.

BL_CONTROL

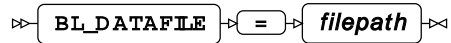
Specifies the path to the SQL*Loader control file.



The control file is used to specify how data is loaded from the data file specified in `BL_DATAFILE` into the database table. The file has the extension or type `*.ctrl`.

BL_DATAFILE

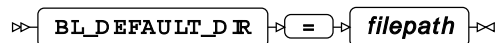
Specifies the path to the SQL*Loader data file.



The data file contains the data to be inserted into the database table. The file has the extension or type `*.dat`.

BL_DEFAULT_DIR

Specifies the default path to use for the files automatically generated by SQL*Loader (bad, data, discard, log, and parameter).

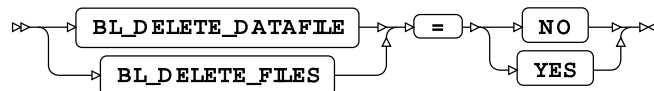


Type: String

The data file contains the data to be inserted into the database table. The file has the extension or type `*.dat`.

BL_DELETE_DATAFILE

Specifies whether the SQL*Loader data and associated control and log files are deleted.



NO

Keep the data, control, and log files after insert.

YES

Delete the data, control, and log files after insert.

BL_DELETE_ONLY_DATAFILE

Specifies whether only the SQL*Loader data file is deleted.



NO

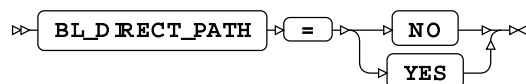
Keep the data after insert. The control and log files are also kept.

YES

Delete only the data file after insert.

BL_DIRECT_PATH

Specifies that the direct path load is used, rather than a conventional path load, to bulk insert data.



NO

Use a conventional path load.

YES

Use a direct path load. This method can be used with either the Oracle Call Interface or SQL*Loader. To use it with SQL*Loader, also specify `BL_USE_SQLLDR=YES`.

BL_DISCARDFILE

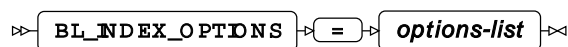
Specifies the path to the SQL*Loader discard file.



The discard file contains data that has neither been inserted into the database table, nor been rejected as bad after the bulk insert process. The file has the extension or type `*.dsc`.

BL_INDEX_OPTIONS

Specifies how indexes are created when inserting information through SQL*Loader.



Type: String

SORTEDINDEX

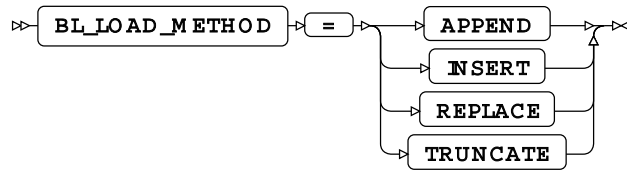
Inserts an index entry at the same time as a row is inserted into a table.

SINGLEROW

Adds data to an empty table or appends data to a table that already contains rows.

BL_LOAD_METHOD

Specifies the method used by SQL*Loader to bulk insert data into a database table.

**APPEND**

Add data to an empty table or append data to a table that already contains rows.

INSERT

Add data to an empty table. Attempting to insert data into a non-empty table results in an error.

REPLACE

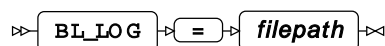
Replace all existing content in the database table with the content of the SQL*Loader data file. An SQL `DELETE FROM TABLE...` statement is first executed, and any delete triggers created on the table are therefore run before the new data is inserted.

TRUNCATE

Replace all existing content in the database table with the content of the SQL*Loader data file. A `TRUNCATE TABLE...` statement is executed as the first step to reset the number of table rows to zero. Any referential constraints on the table must be disabled before using this load method.

BL_LOG

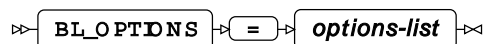
Specifies the path to the SQL*Loader log file.



The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type `*.log`.

BL_OPTIONS

Specifies options for SQL*Loader.

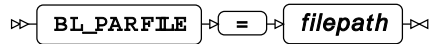


Type: String

The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type `*.log`.

BL_PARFILE

Specifies the path to the SQL*Loader parameter file.



This file is used to store frequently-used command line options. The file has the extension or type *.par.

BL_PRESERVE_BLANKS

Specifies whether SQL*Loader trims trailing spaces from inserted data.



NO

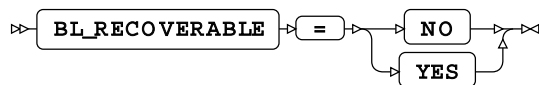
Remove trailing spaces.

YES

Do not remove trailing spaces.

BL_RECOVERABLE

Attempt to recover from an error. The data is added to the redo log, and this log is used as the basis of the recovery.



NO

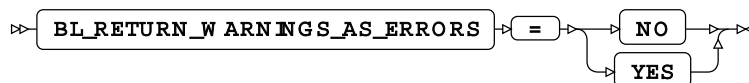
Do not attempt to recover from errors. No redo log is created.

YES

Attempt to recover from an error. The data is added to the redo log, and this log is used as the basis of the recovery.

BL_RETURN_WARNINGS_AS_ERRORS

Specifies whether warnings generated when bulk-inserting data with SQL*Loader are displayed as errors in the SQL*Loader log file.



NO

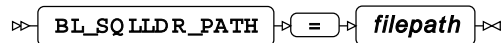
Do not display warnings as errors in the SQL*Loader log file.

YES

Display warnings as errors in the SQL*Loader log file.

BL_SQLLDR_PATH

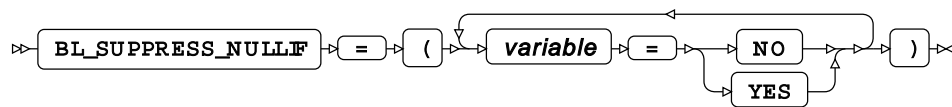
Specifies the path of the SQL*Loader application.



Type: String

BL_SUPPRESS_NULLIF

Specifies whether to ignore NULLIF conditions defined in the SQL*Loader control file.

**variable**

The name of a column (variable), or `_ALL_`.

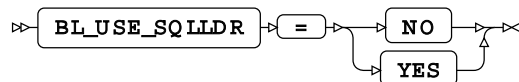
Specify:

- `variable=YES` to ignore the NULLIF condition for the column *variable*
- `variable=NO` to ignore the NULLIF condition for the column *variable*; this is the default.
- `_ALL_=YES` to ignore NULLIF conditions for all columns,

If you specify `_ALL_=YES`, do not specify subsequent arguments. If you do, the list of arguments might not execute correctly.

BL_USE_SQLLDR

Specifies whether bulk inserts are made using Oracle's SQL*Loader.

**Note:**

SQL*Loader is not included in the Oracle Instant Client and must be installed separately.

NO

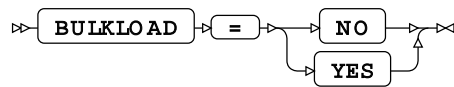
Do not use SQL*Loader for bulk inserts; the Oracle Call Interface (OCI) is used instead.

YES

Use SQL*Loader for bulk inserts.

BULKLOAD

Specifies whether the library connection permits bulk insert into a database table.



Default value: FALSE

NO

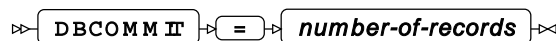
Data cannot be bulk inserted. All other bulkload options are ignored.

YES

Data can be bulk inserted.

DBCMMIT

Specifies the maximum number of records in each commit that is passed to the database server.



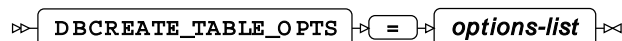
Type: Numeric

Minimum value: 0

Any value entered is accepted, but ignored.

DBCREATE_TABLE_OPTS

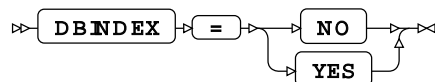
Specifies extra table options to be added to a `CREATE TABLE...` statement after the columns in the table have been defined.



Type: String

DBINDEX

Specifies whether database indexes are used, where available, to provide a fast table look-up when joining a table with a WPS dataset. Database indexes are only used in this way when an available index has the same fields as the join keys.



Default value: FALSE

Specifies whether database indexes are used, where available, to provide a fast table look-up when joining a table with a WPS dataset. Database indexes are only used in this way when an available index has the same fields as the join keys.

NO

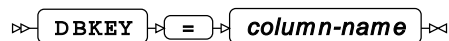
Do not use database indexes.

YES

Use database indexes.

DBKEY

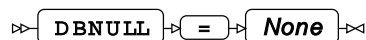
Specifies a table column to use as a database key.



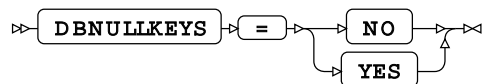
The key is used to index tables for faster access, and for matching rows in datasets.

DBNULL

Specifies that a missing value in a dataset is written to a table as a null.

**DBNULLKEYS**

Specifies what should happen if a value in a column specified as a key using **DBKEY** is null.



Default value: TRUE

DBKEY is a dataset option.

NO

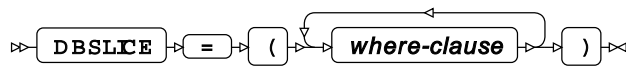
Specifies that if the value in a column defined as a key column using **DBKEY** is null, the row that contains the null value is ignored.

YES

Specifies that if the value in a column defined as a key column using **DBKEY** is null, the row that contains the null value is processed.

DBSLICE

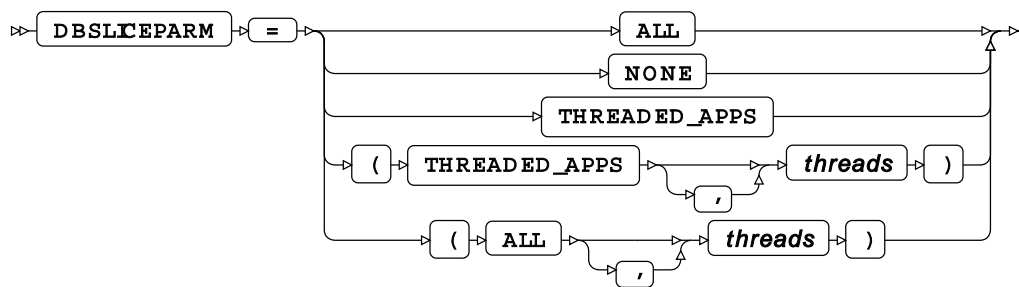
Enables you to divide large tables into smaller tables when you perform operations on the table. The table is divided using a condition.



This option can be used to improve performance when performing operations on large tables. The table is divided by a condition specified in *where-clause*. You can specify more than one *where-clause*. When the table has been divided into smaller tables, operations are performed on the smaller tables. A separate thread is used for the operations on each of the smaller tables.

DBSLICEPARM

Specifies whether to perform multi-threaded reads, in which a part of the database table is returned on each thread.



This option can be used to improve the performance of read operations on large database tables. You can specify a number of threads to be used; different parts of the table are then returned on each thread.

Note:

The default value is that set by the system option `DBSLICEPARM`. The default if the system option has not been set is `THREADED_APPS, 2`.

ALL

The number of available threads is equal to the number of available CPUs. To limit the number of threads used, specify the number in *threads*.

NONE

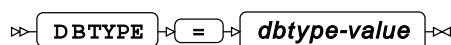
Multiple threads are not used. Data is returned on one thread.

THREADED_APPS

The number of available threads is equal to the number of available CPUs. To limit the number of threads used, specify the number in *threads*.

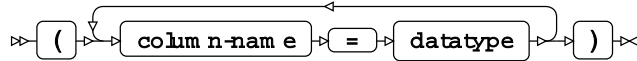
DBTYPE

Specifies data types for columns rather than using mapped data types.



This option enables you to change the data type defined by default for a column when you create a table from a WPS dataset.

dbtype-value has the format:



where *column-name* is the name of a database column, and *datatype* is a database data type.

You can redefine multiple columns; separate each column name definition with spaces. If you specify the same column name more than once, the last definition is applied.

For example:

```
LIBNAME Mylib ORACLE path='oracle-xe-el7/XE' user=xxxx password=xxxxxxxx;
DATA MyLib.a_xyz (dbtype = ( x = long y = integer ));
  SET Mynums;
RUN;
```

By default, numbers in a dataset are mapped to Oracle table as the `NUMBER` data type. `DBTYPE` has been set in this program to specify that the variables `x` and `y` in the dataset are written to column `x` with an Oracle data type `LONG` and `y` with a data type `NUMBER (38)` (the Oracle equivalent of `INTEGER`).

INSERTBUFF

Specifies the number of records to be inserted into the database.



Type: Numeric

Minimum value: 1

Default value: 10

If the `DBCOMMIT` option is specified on the connection statement, the *buffer-size* is the lower value of either `INSERTBUFF` or `DBCOMMIT`. For example if `INSERTBUFF = 100`, and `DBCOMMIT = 200`, then 100 is selected.

PRESERVE_COL_NAMES

Specifies whether the case of variable names in datasets is preserved in column names in tables.



Default value: FALSE

This option enables you to specify how dataset variable names specified in lower case or mixed case are handled when written to a database table.

Note:

If column names are in quotation marks, the database always preserves the case.

Any use of the column name in a SAS language program, however it is written – `xX` or `XX` – refers to the same column.

If column names are case-sensitive, a table can contain multiple similarly-named columns; for example, `clientName` and `CLIENTNAME` are treated as different columns.

NO

Case is not preserved. The name reverts to the default case the database.

For example, if a dataset contains a variable labelled `xX`, and the dataset is written to a database table, the column is labelled `XX`.

YES

Case is preserved.

Note:

This option does not function with a case-insensitive database.

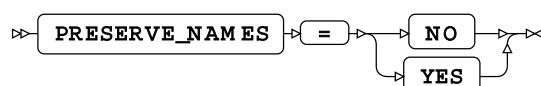
For example, the following program reads data from a table, and writes it to a dataset:

```
DATA MyLib.table1 (PRESERVE_COL_NAMES = YES);  
  SET Mynums;  
  PUT XX;  
RUN;
```

The dataset `Mynums` contains variables named `xX` and `yY`. The resulting table `table1` in the database specified by `MyLib` contains columns with names in the same case. In the `PUT` statement, however, the variable is specified in upper case; because the SAS language ignores the case of variables in statements, the value of the variable `xX` is written to the log.

PRESERVE_NAMES

Specifies whether both the table and column names in SQL statements are treated as case-sensitive or case-insensitive identifiers.



Default value: `FALSE`

NO

Table and column names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example if you have a table labelled `Customer`, this is typically treated as if it is labelled `CUSTOMER`. Any use of the table name, however it is written – `customer` or `Customer` – refers to the same table. The same rule applies to columns in the table; this means the same table and column combination is referenced in a SAS language program whichever case is used in the names.

YES

Table and column names in SQL statements are treated as case sensitive. As part of the statement processing, the names have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test...` becomes `SELECT "test"."myColumn" FROM "test"...`

Using case-sensitive names means that a combination of table and column labels within a SAS language program must accurately reflect the case of the names used within the database. For example, a statement such as `SELECT Test.Column1 FROM Test WHERE TEST.Column2...` would be interpreted as `SELECT "Test"."Column1" FROM "Test" WHERE "TEST"."Column2"...`, meaning the `WHERE...` clause in the in the statement has applied a different table than that in the `SELECT...` clause.

PRESERVE_TAB_NAMES

Specifies whether the table names in SQL statements are treated as case-sensitive – and converted to the normal label form for the database – or case-insensitive identifiers.



Default value: NO

NO

Table names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example, if you have a table labelled `myTable`, this is treated as if it is labelled `MYTABLE`. Any use of the table name in a SAS language program, however it is written – `MyTable` or `mytable` – refers to the same table.

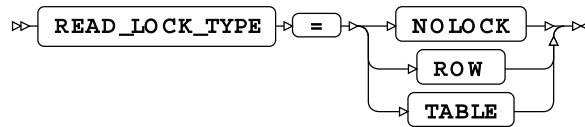
YES

Table names in SQL statements are treated as case sensitive. When a statement is processed, names are have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test` is interpreted as `SELECT "test".MYCOLUMN FROM "test"`.

Using case-sensitive names enables a database to contain multiple, similarly-named tables, because `clientName` and `CLIENTNAME` are treated as different tables.

READ_LOCK_TYPE

Specifies the type of lock to apply to read actions on a table.



Default value: NOLOCK

The type of lock applied to read actions on a table.

NOLOCK

No lock is applied to the table. There are no restrictions on updates to the table, and multiple, concurrent read actions of the data might return different results.

ROW

When `LOCKWAIT=NO`, the WPS client obtains a *row share* lock on a row. This lock allows concurrent access, but prevents a full table lock. This lock also prevents updates to the row until the read action is complete.

TABLE

When `LOCKWAIT=NO`, WPS obtains a *share* lock on the table. This lock allows concurrent read access, but prevents any updates to the table until the query action is complete.

READBUFF

Specifies the number of records retrieved from the database in a single read action.

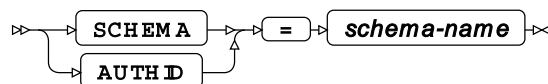


Type: Numeric

Minimum value: 1

SCHEMA

Specifies the name of the database schema with which the connection interacts.

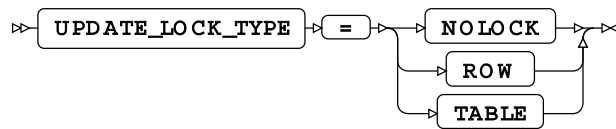


Type: String

The schema is a grouping of database objects and data accessible by the user that can be manipulated through SQL statements.

UPDATE_LOCK_TYPE

Specifies the type of lock to apply to update actions on a table in the database.



Default value: `NOLOCK`

NOLOCK

The WPS client obtains an *exclusive* lock on the table and makes the required updates. Any other activity on the table is delayed until updates are complete.

ROW

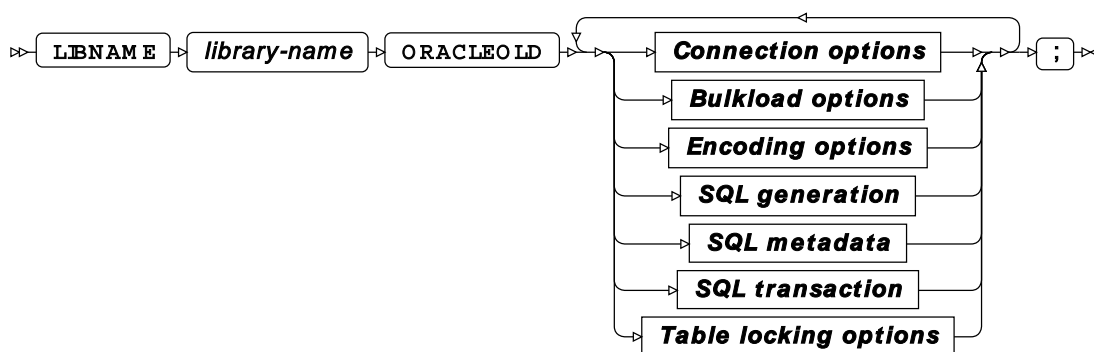
The WPS client obtains a *row share* lock on the table and makes the required updates. This lock allows multiple client access to the table for reading and updating. None of those clients can obtain an *exclusive* lock on the table, and no other client can update the locked row until the transaction is complete.

TABLE

The WPS client obtains an *exclusive* lock on the table and makes the required updates. Any other activity on the table is delayed until updates are complete.

ORACLEOLD

The `LIBNAME` library connection statement provides access to an Oracle database using the single-threaded engine.



The `LIBNAME` library reference enables a program to access to the database using the name defined in the *library-name* argument. You can use the specified library reference in any SAS language programs to access data stored in the database, as long as programs are run in the same WPS session as that in which the library reference was specified. The library reference is only active during the current WPS session. The `LIBNAME` statement contains options that, when specified, determine how SAS language programs interact with the database, grouped as follows:

- Connection options [↗](#) (page 3694): Connect to the Oracle database.
- Bulkload options [↗](#) (page 3698): Rapidly insert large amounts of data into a database using bulk loading (bulk insert).
- Encoding options [↗](#) (page 3706): Manage the encoding differences between database and WPS client.
- SQL generation options [↗](#) (page 3708): Affect how SQL statements are created, and whether the statements are processed by the database or WPS.
- SQL metadata options [↗](#) (page 3711): Determine how table description information or query statements are formatted and used.
- SQL transaction options [↗](#) (page 3714): Affect how SQL statements consisting of execution and data integrity are passed between the Oracle server and WPS.
- Table locking options [↗](#) (page 3715): Determine how WPS interacts with the Oracle table and row locking mechanisms.

In addition to options specified using the library connection statement, options can be specified for individual datasets. When dataset options are specified, these options override the option set using the library connection statement. For more information, see [ORACLE dataset options](#) [↗](#) (page 3680).

library-name

The name used in other SAS language statements to access the database.

For example, the following statement:

```
LIBNAME myLib ORACLEOLD PATH='TNS-name' USER=user-name  
PASSWORD password;
```

Note:

The Transparent Network Substrate name (*TNS-name*) is created on a computer when an Oracle server exists on a remote site. The *TNS-name* enables an ODBC connection between client and server.

creates a connection to a database using the name `MyLib`. This name can then be used in, for example, the `SQL` procedure:

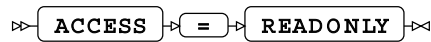
```
PROC SQL;  
  INSERT INTO MyLib.person VALUES (32, 'Smith', 'John', 479216691);  
QUIT;
```

Connection options

Connect to the Oracle database.

ACCESS

Specifies the access mode for the library connection.

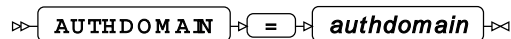


READONLY

The library connection can only be used to read data. Specifying `ACCESS = READONLY` overrides insert or update settings in other options and can result in data not being modified as expected.

AUTHDOMAIN

Specifies the authorisation domain.



Type: String

The authorisation domain provides permissions to access a database server. WPS uses Hub as an authorisation domain, and a Hub server must be available to your system.

In this example, permissions for accessing the Hub are supplied as system options, and the name of the authorisation domain containing the authorisation details in the Hub is specified to AUTHDOMAIN.

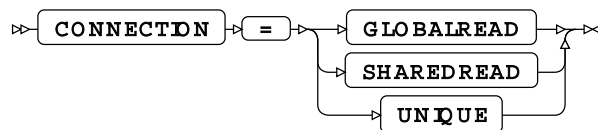
```
OPTIONS HUB_SERVER='blue_streak' HUB_PORT=309 HUB_PROTOCOL='HTTP'  
HUB_USER='ARichards' HUB_PWD='*****';  
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' AUTHDOMAIN='OracleAuth';
```

Note:

If `USER` and `PASSWORD` are specified in the `LIBNAME` statement, then `AUTHDOMAIN` is ignored.

CONNECTION

Specifies the library connection type to the database when not in single-connection mode.



Default value: SHAREDREAD

Note:

- If a database can only accept one SQL statement per connection, the connection type is always treated as a `UNIQUE` connection, whatever value is specified to this option.
- This option is ignored on z/OS systems, where the connection type is always treated as unique.

GLOBALREAD

A single connection to a database is created for this library connection statement, and for any other libraries that match the connection options. This connection to the database is used for all read operations. However, a separate connection to the database is created for each write operation.

SHAREDREAD

All database read operations that use this library connection statement share the same connection to the database that is created when the library connection statement is invoked. A connection to the database is created when the first insert or update operation occurs, and then this connection is shared for subsequent updates to the database.

This option interacts with the `UTILCONN_TRANSIENT` option as follows:

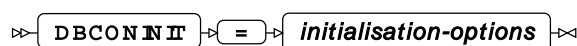
- If `UTILCONN_TRANSIENT = YES`, a utility connection is created each time the connection to the database is required, but not persisted for future connections.
- If a utility connection does not exist, and `UTILCONN_TRANSIENT = NO`, a utility connection is created and stored for future connections to the database.
- If a utility connection already exists, and `UTILCONN_TRANSIENT = NO` is specified, the existing utility connection is used and then stored for future connections to the database.

UNIQUE

Each operation using this library connection statement has its own connection to the database.

DBCONINIT

Specifies SQL statements or database commands that are processed every time the library connection is opened.



Type: String

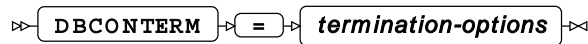
The connection is opened when this `LIBNAME` statement is first executed, and then every time a connection to the database is made; for example, by running an SQL statement in the SQL procedure.

In this example, a connection is created to an Oracle database. An existing table `UK_PMJan` is deleted, if it exists:

```
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' USER=test PASSWORD=*****  
DBCONINIT='DROP TABLE IF EXISTS UK_PMJan';
```

DBCONTERM

Specifies SQL statements or database commands that are processed every time the library connection is closed.



Type: String

In this example, a new table UK_PMFeb is created and the table UK_PMJJan is deleted, if it exists.

```

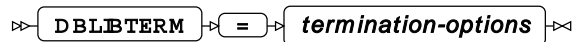
LIBNAME MyLib ORACLE PATH='oracle-pz-e27/ZY' USER=test
PASSWORD=***** DBCONTERM='DROP TABLE IF EXISTS UK_PMJJan';

DATA myLib.UK_PMFeb;
SET UK_PM;

RUN;
  
```

DBLIBTERM

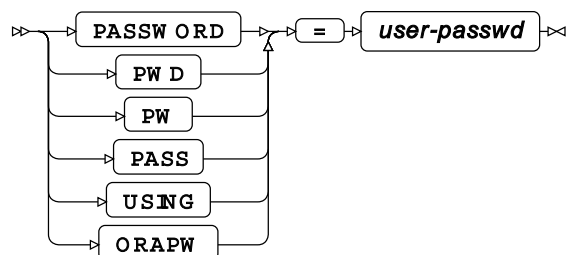
Specifies SQL statements or database commands that are processed after the first library connection has been successfully made.



Type: String

PASSWORD

Specifies the password for the user name.

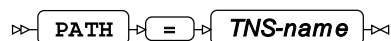


Type: String

If special characters are used as part of the string, you must enter *user-passwd* in quotation marks. The user name is specified with the `USER` option.

PATH

Specifies the TNS service name for the Oracle database.

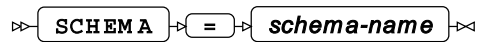


Type: String

The TNS name is an alias for the Oracle Call Interface (OCI) connection string that identifies the database server and instance to which you want to connect. For example: `PATH= 'oracle-pg-e27/ZY'`.

SCHEMA

Specifies the name of the database schema with which the connection interacts.

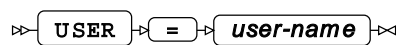


Type: String

The schema is a grouping of database objects and data accessible by the user that can be manipulated through SQL statements.

USER

Specifies the user name required to access the database.

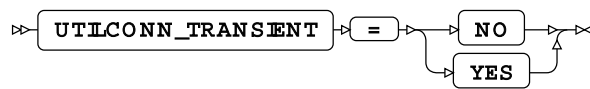


Type: String

If special characters are used as part of the string, you must enter *user-name* in quotation marks.

UTILCONN_TRANSIENT

Specifies whether the *utility connection* is transient or non-transient.



A utility connection is created to separate data read and update processes from metadata handling. This connection is used by WPS Workbench to list the members of the database in which to populate information; for example, the *Sashelp* tables.

NO

A utility connection is created to separate data read and update processes from metadata handling. This connection is used by WPS Workbench to list the members of the database in which to populate information; for example, the *Sashelp* tables.

YES

The utility connection is created whenever required, but is not persisted for future connections to the database.

Bulkload options

Rapidly insert large amounts of data into a database using bulk loading (bulk insert). Bulk insert uses either the native Oracle Call Interface (OCI) or, if installed, SQL*Loader.

SQL*Loader provides a bulk insert mechanism that can be faster than the OCI, because indexes are only updated when data insertion has finished.

Note:

With SQL*Loader you can inadvertently insert incorrect data into an index, and create a table that cannot be accessed. You should therefore keep the intermediate data file that enables you to validate information on insertion.

Oracle supports two methods for bulk loading data:

- Conventional path load. SQL `INSERT` statements are constructed that encapsulate the data to be inserted. This is the default method used by both OCI or SQL*Loader.
- Direct path load. Data is bulk loaded directly using the OCI direct path API.

Library connection options enable you to specify which bulk loading methods to use. To bulk load data using:

- The conventional path and the OCI, specify `BULKLOAD=YES`.
- The conventional path and SQL*Loader, specify:

```
BULKLOAD=YES BL_USE_SQLLDR=YES
```

- The direct path and the OCI, specify:

```
BULKLOAD=YES BL_DIRECT_PATH=YES
```

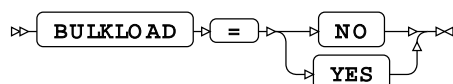
- The direct path and SQL*Loader, specify:

```
BULKLOAD=YES BL_USE_SQLLDR=YES BL_DIRECT_PATH=YES
```

For information on these options, see [BULKLOAD](#) (page 3699), [BL_DIRECT_PATH](#) (page 3701) [BL_USE_SQLLDR](#) (page 3705).

BULKLOAD

Specifies whether the library connection permits bulk insert into a database table.



Default value: NO

NO

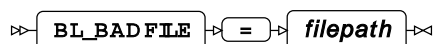
Data cannot be bulk inserted. All other bulkload options are ignored.

YES

Data can be bulk inserted.

BL_BADFILE

Specifies the path to the SQL*Loader bad file.



Type: String

The bad file is used to store records that could not be inserted into the database table due to errors. The file is only created if required. The file has the extension or type *.bad.

BL_CONTROL

Specifies the path to the SQL*Loader control file.

⇒ **BL_CONTROL** ⇒ = ⇒ *filepath* ⇒

Type: String

The control file is used to specify how data is loaded from the data file specified in **BL_DATAFILE** into the database table. The file has the extension or type *.ctrl.

BL_DATAFILE

Specifies the path to the SQL*Loader data file.

⇒ **BL_DATAFILE** ⇒ = ⇒ *filepath* ⇒

Type: String

The data file contains the data to be inserted into the database table. The file has the extension or type *.dat.

BL_DATECACHE_SIZE

Specifies the size of the date cache to use with a bulk insert using OCI.

⇒ **BL_DATECACHE_SIZE** ⇒ = ⇒ *buffer-size* ⇒

Type: Numeric

Minimum value: 0

Default value: 65536

This cache is used when data is inserted that contains date information. A date value needs to be converted from a SAS language date format to an Oracle data type before it is inserted into a table. A converted date value is stored in this cache to speed insertion of duplicate information.

BL_DEFAULT_DIR

Specifies the default path to use for the files automatically generated by SQL*Loader (bad, data, discard, log, and parameter).

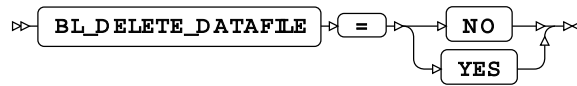
⇒ **BL_DEFAULT_DIR** ⇒ = ⇒ *filepath* ⇒

Type: String

If pathnames for these automatically generated files are not specified using the corresponding bulkload options, a file is automatically created when required and stored in the location specified by this option.

BL_DELETE_DATAFILE

Specifies whether the SQL*Loader data and associated control and log files are deleted.



Default value: YES

NO

Keep the data, control, and log files after insert.

YES

Delete the data, control, and log files after insert.

BL_DELETE_ONLY_DATAFILE

Specifies whether only the SQL*Loader data file is deleted.



Default value: NO

NO

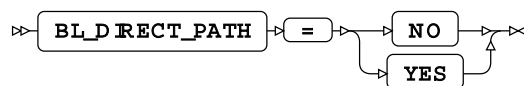
Keep the data after insert. The control and log files are also kept.

YES

Delete only the data file after insert.

BL_DIRECT_PATH

Specifies that the direct path load is used, rather than a conventional path load, to bulk insert data.



Default value: NO

NO

Use a conventional path load.

YES

Use a direct path load. This method can be used with either the Oracle Call Interface or SQL*Loader. To use it with SQL*Loader, also specify `BL_USE_SQLLDR=YES`.

BL_DISCARDFILE

Specifies the path to the SQL*Loader discard file.



Type: String

The discard file contains data that has neither been inserted into the database table, nor been rejected as bad after the bulk insert process. The file has the extension or type `*.dsc`.

BL_INDEX_OPTIONS

Specifies how indexes are created when inserting information through SQL*Loader.



Type: String

SORTEDINDEX

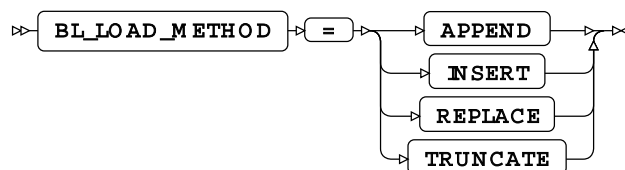
Inserts an index entry at the same time as a row is inserted into a table.

SINGLEROW

Adds data to an empty table or appends data to a table that already contains rows.

BL_LOAD_METHOD

Specifies the method used by SQL*Loader to bulk insert data into a database table.



Default value: APPEND

APPEND

Add data to an empty table or append data to a table that already contains rows.

INSERT

Add data to an empty table. Attempting to insert data into a non-empty table will result in an error.

REPLACE

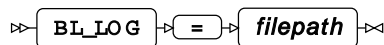
Replace all existing content in the database table with the content of the SQL*Loader data file. An SQL `DELETE FROM TABLE...` statement is first executed, and any delete triggers created on the table are therefore run before the new data is inserted.

TRUNCATE

Replace all existing content in the database table with the content of the SQL*Loader data file. A `TRUNCATE TABLE...` statement is executed as the first step to reset the number of table rows to zero. Any referential constraints on the table must be disabled before using this load method.

BL_LOG

Specifies the path to the SQL*Loader log file.

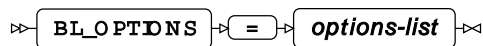


Type: String

The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type `*.log`.

BL_OPTIONS

Specifies options for SQL*Loader.

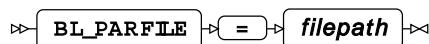


Type: String

Where these options are specified in both the SQL*Loader control file and `BL_OPTIONS`, any options specified using `BL_OPTIONS` take priority.

BL_PARFILE

Specifies the path to the SQL*Loader parameter file.



Type: String

The path to the SQL*Loader parameter (`*.par`) file, used to store frequently-used command line options.

BL_PRESERVE_BLANKS

Specifies whether SQL*Loader trims trailing spaces from inserted data.



Default value: YES

NO

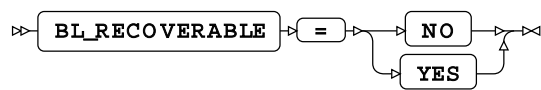
Remove trailing spaces.

YES

Do not remove trailing spaces.

BL_RECOVERABLE

Specifies whether the SQL*Loader bulk insert attempts to reload data if there is a problem during the insert that is not related to data.



Default value: NO

NO

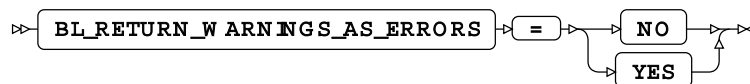
Do not attempt to recover from errors. No redo log is created.

YES

Attempt to recover from an error. The data is added to the redo log, and this log is used as the basis of the recovery.

BL_RETURN_WARNINGS_AS_ERRORS

Specifies whether warnings generated when bulk-inserting data with SQL*Loader are displayed as errors in the SQL*Loader log file.



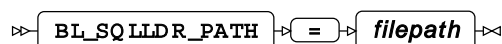
NO

Do not display warnings as errors in the SQL*Loader log file.

YES

Display warnings as errors in the SQL*Loader log file.

BL_SQLLDR_PATH

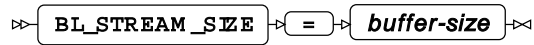


Type: String

Specifies the path of the SQL*Loader (.sqlldr) application.

BL_STREAM_SIZE

Specifies the path of the SQL*Loader application.

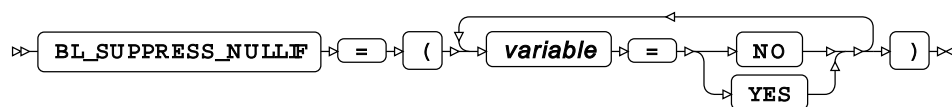


Type: Numeric

Minimum value: 0

BL_SUPPRESS_NULLIF

Specifies whether to ignore NULLIF conditions defined in the SQL*Loader control file.



variable

The name of a column (variable), or `_ALL_`.

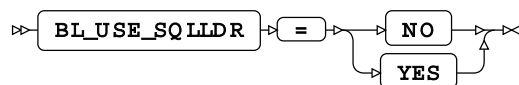
Specify:

- `variable=YES` to ignore the NULLIF condition for the column `variable`
- `variable=NO` to ignore the NULLIF condition for the column `variable`; this is the default.
- `_ALL_=YES` to ignore NULLIF conditions for all columns,

If you specify `_ALL_=YES`, do not specify subsequent arguments. If you do, the list of arguments might not execute correctly.

BL_USE_SQLLDR

Specifies whether bulk inserts are made using Oracle's SQL*Loader.



Default value: NO

Note:

SQL*Loader is not included in the Oracle Instant Client and must be installed separately.

NO

Do not use SQL*Loader for bulk inserts; the Oracle Call Interface (OCI) is used instead.

YES

Use SQL*Loader for bulk inserts.

Encoding options

Manage the encoding differences between database and WPS client.

ADJUST_BYTE_SEMANTIC_COLUMN_LENGTHS

Specifies how to handle database columns defined as `CHAR` and `VARCHAR` when WPS and database server encodings differ.



If the encodings between the database server and WPS differ, and database columns have been specified as the `CHAR` and `VARCHAR` client column size is adjusted to store all characters retrieved from the database server.

The default depends on the value of `DBCLIENT_MAX_BYTES`. If `DBCLIENT_MAX_BYTES` is equal to 1, then the default is `NO`. If `DBCLIENT_MAX_BYTES` is greater than 1, then the default is `YES`.

NO

The WPS variable holds the same number of bytes as the corresponding column in the database. If the number of bytes used to represent a character differs between the character sets used by WPS and by the database, data might be truncated.

YES

Adjust variable lengths in WPS to ensure all characters from the corresponding column are accurately represented.

For example, if the character set on the database server uses two bytes to represent a character, a 500 character column is returned as 1000 bytes. If the character set in WPS uses four bytes to represent a character, the client reads the number of characters ($1000/2 = 500$ characters) and allocates the correct amount of space ($500*4=2000$ bytes) to ensure the characters are correctly encoded.

ADJUST_NCHAR_COLUMN_LENGTHS

Specifies how to handle database columns defined as `NCHAR` and `NVARCHAR`.



Default value: YES

NO

The WPS variable holds the same number of bytes as the corresponding column in the database. If the number of bytes used to represent a character differs between the character sets used by WPS and by the database, data might be truncated.

YES

The column length in WPS is adjusted to cater for the value defined in `DBSERVER_MAX_BYTES`. The adjustment is calculated using:

$$\text{number_of_characters} * \text{value_of_DBCLIENT_MAX_BYTES}$$

where *number_of_characters* is the number of characters in the column, and *value_of_DBCLIENT_MAX_BYTES*

where *number_of_characters* is the number of characters in the column, and *value_of_DBCLIENT_MAX_BYTES* is the value specified by `DBCLIENT_MAX_BYTES` [\(page 3667\)](#).

DB_LENGTH_SEMANTICS_BYTE

Specifies how to handle database columns defined as `NCHAR` and `NVARCHAR`.



Default value: `TRUE`

NO

Definitions for `CHAR` or `VARCHAR2` columns are created by specifying the total number of characters.

For example, if a dataset has a column that has 20 bytes available to store characters, and each character is composed of:

- One byte, then 20 characters can be stored in each column.
- Two bytes, then only 10 characters can be stored in each column.
- Four bytes, then only five characters can be stored in each column.

The length defined for `CHAR` or `VARCHAR2` allows a maximum number of characters to be stored whatever the character set encoding used on the database. For example, if the character set consists of a fixed two-byte encoding, and the variable is 20 bytes long, then the data type for the variable is the length of the variable, divided by two bytes per character. The data type for the corresponding column is therefore `VARCHAR2 (10 CHAR)`. If a character set uses a variable number of bytes to define a character, then the maximum number of bytes used to define a character is used to calculate the data type; for example, characters in the UTF-8 character set can be represented using up to four bytes, therefore a 20 byte variable is defined as `VARCHAR2 (5 CHAR)` in the database.

YES

Definitions for `CHAR` or `VARCHAR2` data types are created by specifying the data type in bytes. The length of the data type is dependent on the encoding used by the database server, and whether characters have fixed or variable encodings. It is calculated as:

$$\text{number_of_characters} * \text{value_of_DBSERVER_MAX_BYTES}$$

where *number_of_characters* is the number of characters defined for the variable, and *value_of_DBSERVER_MAX_BYTES* is the value specified by `DBSERVER_MAX_BYTES` [\(page 3668\)](#).

For example, assume a dataset has a column of 20 bytes available to store characters. The dataset variable is defined as 10 characters long, and the database character set has four bytes UTF-8 encoding, as specified in `DBSERVER_MAX_BYTES`). The equivalent table column data type is defined as `VARCHAR(40)` (that is, 10 x 4).

DBCLIENT_MAX_BYTES

Specifies the maximum number of bytes used to encode each character in WPS.

⇒ `DBCLIENT_MAX_BYTES` ⇒ `=` ⇒ *bytes-per-character* ⇒

Type: Numeric

Minimum value: 1

When not set, this defaults to the number of bytes used by the current WPS session encoding.

DBSERVER_MAX_BYTES

Specifies the maximum number of bytes per character in the encoding used by the database.

⇒ `DBSERVER_MAX_BYTES` ⇒ `=` ⇒ *bytes-per-character* ⇒

Type: Numeric

Minimum value: 1

SQL generation

Affect how SQL statements are created, and whether the statements are processed by the database or WPS.

DBCREATE_TABLE_OPTS

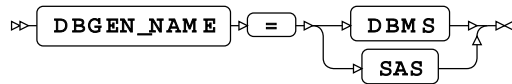
Specifies extra table options to be added to a `CREATE TABLE...` statement after the columns in the table have been defined.

⇒ `DBCREATE_TABLE_OPTS` ⇒ `=` ⇒ *table-options* ⇒

Type: String

DBGEN_NAME

Specifies how to modify column names that require adjustment to be valid in the language of SAS.



Default value: SAS

DBMS

If the column name contains invalid characters, those characters are replaced with an underscore. If this change results in a name that clashes with that of another column, the column count is appended to the column name.

For example, if your table contains the columns `id$x`, `id#x` and `id_x`, the variables in the language of SAS would be `ID_X`, `ID_X0`, and `ID_X1` respectively.

Note:

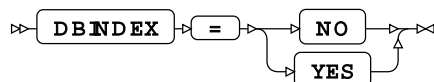
The numbering starts at the first repeat of the variable name, and also begins at 0.

SAS

If the column name contains invalid characters, or have the same name, the column name is replaced with the string `_COL`. If this change results in a name clash with other columns, the column count is appended to the name: `_COLx` where `x` is the column, starting at 0 (zero).

DBINDEX

Specifies whether database indexes are used, where available, to provide a fast table look-up when joining a table with a WPS dataset. Database indexes are only used in this way when an available index has the same fields as the join keys.



NO

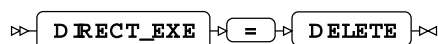
Do not use database indexes.

YES

Enable this option.

DIRECT_EXE

Specifies whether delete statements are executed directly on the database, or through WPS.



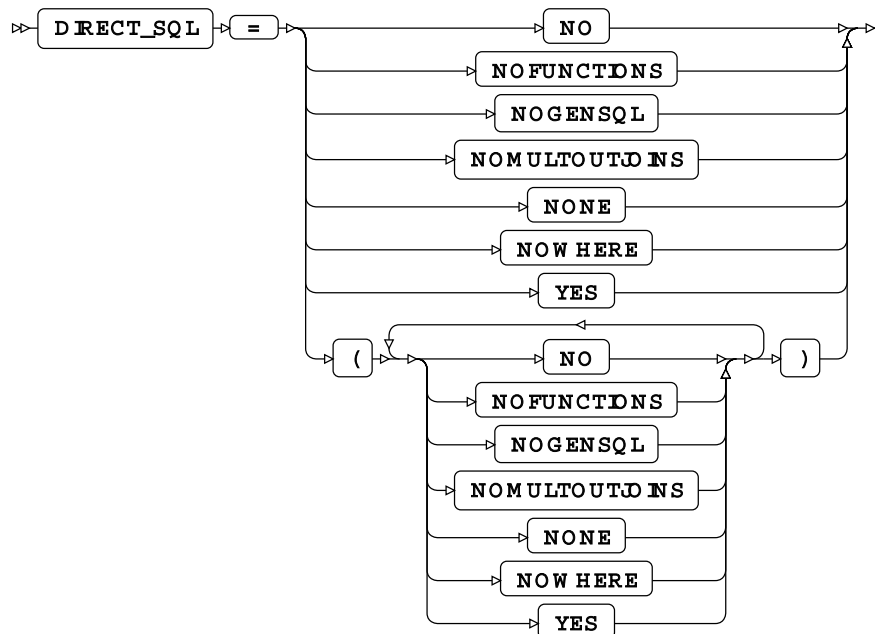
When not specified, the database table is scanned and rows matching the delete criteria are returned to WPS. An SQL `DELETE FROM...` statement is created by WPS for each affected row and passed to the database for processing.

DELETE

The SQL `DELETE FROM...` command is passed to the database and processed entirely by the database engine. In this case, the SQL `DELETE...` statements are not created by WPS.

DIRECT_SQL

Specifies whether SQL statements are passed through for processing by the database server, or processed by WPS.



Default value: YES

NO

All SQL statements are processed by WPS.

NOFUNCTIONS

Any SQL statements containing function calls are processed by WPS. All other statements that can be processed by the database server are passed through to the server for processing.

NOGENSQL

Setting this value has no effect.

NOMULTOUTJOINS

Any SQL statements containing multiple outer joins are processed by WPS. All other statements that can be processed by the database server are passed through to the server for processing.

NONE

All SQL statements are processed by WPS.

NOWHERE

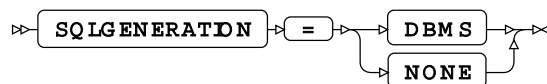
Any statements containing `WHERE` clauses are processed by WPS. All other statements that can be processed by the database server are passed through to the database server for processing.

YES

All SQL statements that can be processed in-database are passed through to the database server for processing.

SQLGENERATION

Specifies where the SQL statements required to create summary data for the `SUMMARY` procedure are processed.



DBMS

Data processing is carried out by the database server.

NONE

WPS generates the SQL statements and processes data, sending the result to the database server.

SQL metadata

Determine how table description information or query statements are formatted and used.

PRESERVE_COL_NAMES

Specifies whether the case of variable names in datasets is preserved in column names in tables.



Default value: NO

Note:

When column names are enclosed in quotes, the database preserves the case. However when column names are not quoted the database examines the content and makes a decision.

NO

Case is not preserved. The name reverts to the default case the database.

For example, if a dataset contains a variable labelled `xx`, and the dataset is written to a database table, the column is labelled `xx`.

YES

Case is preserved.

Note:

This option does not function with a case-insensitive database.

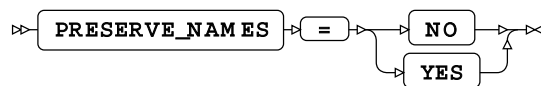
For example, the following program reads data from a table, and writes it to a dataset:

```
DATA MyLib.table1 (PRESERVE_COL_NAMES = YES);  
  SET Mynums;  
  PUT XX;  
RUN;
```

The dataset `Mynums` contains variables named `xx` and `yy`. The resulting table `table1` in the database specified by `MyLib` contains columns with names in the same case. In the `PUT` statement, however, the variable is specified in upper case; because the SAS language ignores the case of variables in statements, the value of the variable `xx` is written to the log.

PRESERVE_NAMES

Specifies whether both the table and column names in SQL statements are treated as case-sensitive or case-insensitive identifiers.



Default value: NO

NO

Table and column names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example if you have a table labelled `Customer`, this is typically treated as if it is labelled `CUSTOMER`. Any use of the table name, however it is written – `customer` or `Customer` – refers to the same table. The same rule applies to columns in the table; this means the same table and column combination is referenced in a SAS language program whichever case is used in the names.

YES

Table and column names in SQL statements are treated as case sensitive. As part of the statement processing, the names have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test...` becomes `SELECT "test"."myColumn" FROM "test"...`

Using case-sensitive names means that a combination of table and column labels within a SAS language program must accurately reflect the case of the names used within the database. For example, a statement such as `SELECT Test.Column1 FROM Test WHERE TEST.Column2...` would be interpreted as `SELECT "Test"."Column1" FROM "Test" WHERE "TEST"."Column2"...`, meaning the `WHERE...` clause in the `in the` statement has applied a different table than that in the `SELECT...` clause.

PRESERVE_TAB_NAMES

Specifies whether the table names in SQL statements are treated as case-sensitive – and converted to the normal label form for the database – or case-insensitive identifiers.



Default value: NO

NO

Table names in SQL statements are treated as case insensitive and converted to the normal label form – typically uppercase – when the SQL statement is processed.

For example, if you have a table labelled `myTable`, this is treated as if it is labelled `MYTABLE`. Any use of the table name in a SAS language program, however it is written – `MyTable` or `mytable` – refers to the same table.

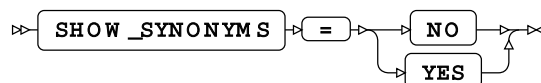
YES

Table names in SQL statements are treated as case sensitive. When a statement is processed, names are have opening quotation marks and closing quotation marks added and used as entered. For example, `SELECT test.myColumn FROM test` is interpreted as `SELECT "test".MYCOLUMN FROM "test"`.

Using case-sensitive names enables a database to contain multiple, similarly-named tables, because `clientName` and `CLIENTNAME` are treated as different tables.

SHOW_SYNONYMS

Specifies whether the Oracle-specific alternative name for database objects is used when querying the database structure.



Default value: NO

NO

The original object names are used when viewing the information about a database through `PROC DATASETS`, or when querying `DICTIONARY` tables through `PROC SQL`.

YES

Created synonyms are used when viewing the information about a database through `PROC DATASETS`, or when querying `DICTIONARY` tables through `PROC SQL`.

SQL transaction

Affect how SQL statements consisting of execution and data integrity are passed between the Oracle server and WPS.

DBCOMMIT

Specifies the number of records that are passed to the Oracle database server before a commit is made.

```
DBCOMMIT = number-of-records
```

Type: Numeric

Minimum value: 0

DBLINK

This option is not supported.

```
DBLINK = link-name
```

Type: String

DBMAX_TEXT

Specifies the maximum string length (number of characters) that can be inserted into a field in a database table.

```
DBMAX_TEXT = max-string-length
```

Type: Numeric

Minimum value: 1

Maximum value: 32767

Default value: 1000

INSERTBUFF

Specifies the number of records to be inserted into the database.

```
INSERTBUFF = buffer-size
```

Type: Numeric

Minimum value: 1

Maximum value: 32767

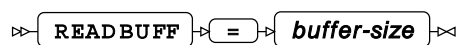
Default value: 10

This number applies if you are not also specifying bulkload options.

If the `DBCOMMIT` option is specified on the connection statement, the *buffer-size* is the lower value of either `INSERTBUFF` or `DBCOMMIT`. For example if `INSERTBUFF` = 100, and `DBCOMMIT` = 200, then 100 is selected.

READBUFF

Specifies the number of records retrieved from the database in a single read action.



Type: Numeric

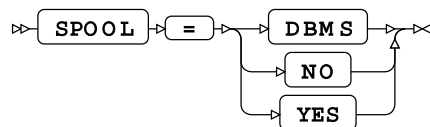
Minimum value: 1

Maximum value: 32767

Default value: 250

SPOOL

Spooling creates a temporary file containing the results of reading a database table.



This cache file is used for subsequent read actions on the data, enabling data to be read more quickly than re-reading the content from the database table.

DBMS

Spooling is active and handled by the DBMS.

NO

Spooling is not active in WPS. Any attempt by WPS to spool will generate an error.

YES

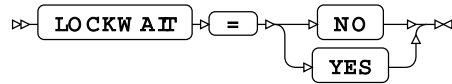
Spooling is active in WPS.

Table locking options

Determine how WPS interacts with the Oracle table and row locking mechanisms.

LOCKWAIT

Specifies what happens when another process attempts to read or update a locked database table.



NO

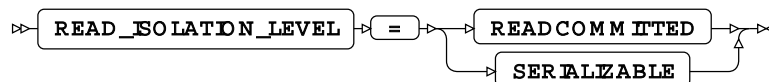
Nothing is read from or written to the table, and a message is written to the log.

YES

Wait for the lock to be removed from the table, and then attempt to read or update the table.

READ_ISOLATION_LEVEL

Specifies when a client can read data that is inserted into a database table if the table is accessed by multiple clients.



Default value: READCOMMITTED

`READ_ISOLATION_LEVEL` settings only affect SQL `SELECT` statements, and then only when the `ACCESS` option is not set to `READONLY`.

READCOMMITTED

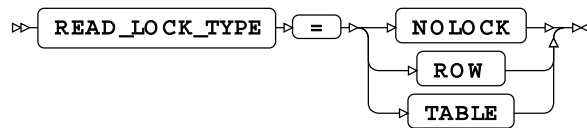
Only committed data is read by a client. Data inserted as part of another client's transaction can be queried once the transaction is complete and the data has been committed to the database. Where multiple read actions take place against a table by the same client, returned results are only updated to reflect any transactional changes made between the read actions.

SERIALIZABLE

Where multiple clients are reading data from the table, only one query is processed at a time – any query of the table data must complete before the next query or data update can begin. If the client attempts to read data from the table while an update transaction is being processed, the client reading the table data cannot access the table until the updating transaction is complete.

READ_LOCK_TYPE

Specifies the type of lock to apply to read actions on a table.



Default value: NOLOCK

NOLOCK

No lock is applied to the table. There are no restrictions on updates to the table, and multiple, concurrent read actions of the data might return different results.

ROW

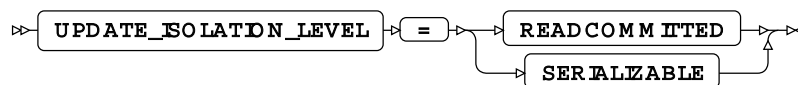
When `LOCKWAIT=NO`, the WPS client obtains a *row share* lock on a row. This lock allows concurrent access, but prevents a full table lock. This lock also prevents updates to the row until the read action is complete.

TABLE

When `LOCKWAIT=NO`, WPS obtains a *share* lock on the table. This lock allows concurrent read access, but prevents any updates to the table until the query action is complete.

UPDATE_ISOLATION_LEVEL

Specifies when a client can update a table if the table is accessed by multiple clients.



Default value: READCOMMITTED

`UPDATE_ISOLATION_LEVEL` settings only affect SQL `UPDATE` statements, and then only when the `ACCESS` option is not set to `READONLY`.

READCOMMITTED

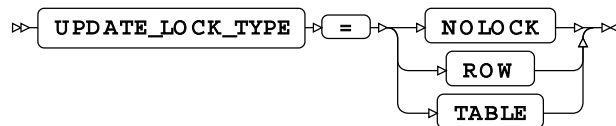
Read before update. A second read might return different data to the original read, as there is no restriction on the update which might insert new rows, delete existing rows, or change data.

SERIALIZABLE

A WPS client waits for the current client to finish its updates to a table before it makes its changes.

UPDATE_LOCK_TYPE

Specifies the type of lock to apply to update actions on a table in the database.



Default value: NOLOCK

NOLOCK

No lock is applied to the table. There are no restrictions on updates to the table, and multiple, concurrent read actions of the data might return different results.

ROW

The WPS client obtains a *row share* lock on the table and makes the required updates. This lock allows multiple client access to the table for reading and updating. None of those clients can obtain an *exclusive* lock on the table, and no other client can update the locked row until the transaction is complete.

TABLE

The WPS client obtains an *exclusive* lock on the table and makes the required updates. Any other activity on the table is delayed until updates are complete.

ORACLEOLD Dataset Options

When dataset options are specified for a database table, the values of those options override the settings specified for the same options on the library connection statement. You should therefore configure a library connection statement so that it provides the best access to all accessible objects in a database. If different options are required for specific tables, the following dataset options can be used.

You can specify dataset options wherever you can specify a database table in a SAS language or SQL statement.

For example, dataset options can be added when a database table is specified in an `SQL INSERT` statement when using the SQL procedure:

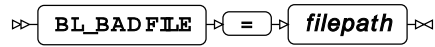
```
PROC SQL;
  INSERT INTO myLib.myTable (BULKLOAD=YES, BL_LOAD_METHOD=TRUNCATE)
    AS SELECT * FROM datasrc.append_src;
QUIT;
```

Similarly, dataset options can be added when a database table is specified as the input dataset in a SAS language program:

```
DATA myoutfile;
  SET mylib.a_xyz1 (BULKLOAD=YES);
RUN;
```

BL_BADFILE

Specifies the path to the SQL*Loader bad file.

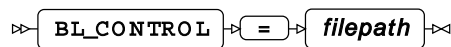


Type: String

The bad file is used to store records that could not be inserted into the database table due to errors. The file is only created if required. The file has the extension or type *.bad.

BL_CONTROL

Specifies the path to the SQL*Loader control file.

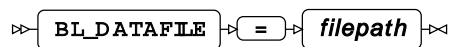


Type: String

The control file is used to specify how data is loaded from the data file specified in BL_DATAFILE into the database table. The file has the extension or type *.ctrl.

BL_DATAFILE

Specifies the path to the SQL*Loader data file.



Type: String

The data file contains the data to be inserted into the database table. The file has the extension or type *.dat.

BL_DEFAULT_DIR

Specifies the default path to use for the files automatically generated by SQL*Loader (bad, data, discard, log, and parameter).

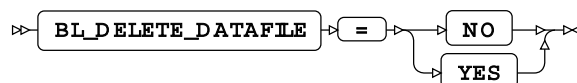


Type: String

The data file contains the data to be inserted into the database table. The file has the extension or type *.dat.

BL_DELETE_DATAFILE

Specifies whether the SQL*Loader data and associated control and log files are deleted.



NO

Keep the data, control, and log files after insert.

YES

BL_DELETE_ONLY_DATAFILE

Delete the data, control, and log files after insert.



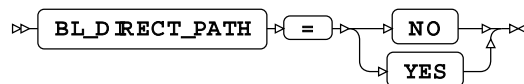
NO

Keep the data after insert. The control and log files are also kept.

YES

BL_DIRECT_PATH

Specifies that the direct path load is used, rather than a conventional path load, to bulk insert data.



NO

Use a conventional path load.

YES

Use a direct path load. This method can be used with either the Oracle Call Interface or SQL*Loader. To use it with SQL*Loader, also specify `BL_USE_SQLLDR=YES`.

BL_DISCARDFILE

Specifies the path to the SQL*Loader discard file.

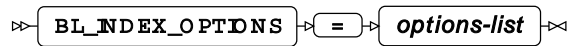


Type: String

The discard file contains data that has neither been inserted into the database table, nor been rejected as bad after the bulk insert process. The file has the extension or type `*.dsc`.

BL_INDEX_OPTIONS

Specifies how indexes are created when inserting information through SQL*Loader.



Type: String

SORTEDINDEX

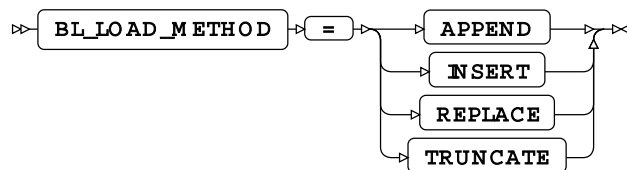
Inserts an index entry at the same time as a row is inserted into a table.

SINGLEROW

Adds data to an empty table or appends data to a table that already contains rows.

BL_LOAD_METHOD

Specifies the method used by SQL*Loader to bulk insert data into a database table.



Default value: APPEND

APPEND

Add data to an empty table or append data to a table that already contains rows.

INSERT

Add data to an empty table. Attempting to insert data into a non-empty table results in an error.

REPLACE

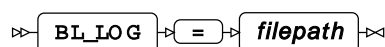
Replace all existing content in the database table with the content of the SQL*Loader data file. An SQL `DELETE FROM TABLE...` statement is first executed, and any delete triggers created on the table are therefore run before the new data is inserted.

TRUNCATE

Replace all existing content in the database table with the content of the SQL*Loader data file. A `TRUNCATE TABLE...` statement is executed as the first step to reset the number of table rows to zero. Any referential constraints on the table must be disabled before using this load method.

BL_LOG

Specifies the path to the SQL*Loader log file.

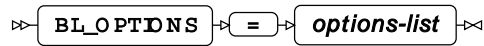


Type: String

The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type *.log.

BL_OPTIONS

Specifies options for SQL*Loader.

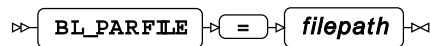


Type: String

The log file is created when the SQL*Loader begins bulk-inserting data, and contains log entries summarising the events during the data insert. The file has the extension or type *.log.

BL_PARFILE

Specifies the path to the SQL*Loader parameter file.



Type: String

BL_PRESERVE_BLANKS

This file is used to store frequently-used command line options. The file has the extension or type *.par.



NO

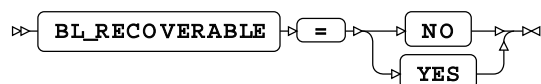
Remove trailing spaces.

YES

Do not remove trailing spaces.

BL_RECOVERABLE

Specifies whether the SQL*Loader bulk insert attempts to reload data if there is a problem during the insert that is not related to data.



NO

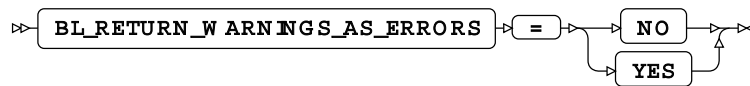
Do not attempt to recover from errors. No redo log is created.

YES

Attempt to recover from an error. The data is added to the redo log, and this log is used as the basis of the recovery.

BL_RETURN_WARNINGS_AS_ERRORS

Specifies whether warnings generated when bulk-inserting data with SQL*Loader are displayed as errors in the SQL*Loader log file.

**NO**

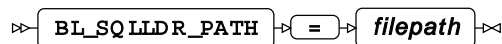
Do not display warnings as errors in the SQL*Loader log file.

YES

Display warnings as errors in the SQL*Loader log file.

BL_SQLLDR_PATH

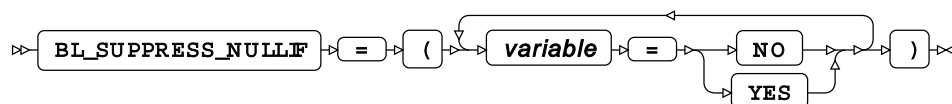
Specifies the path of the SQL*Loader application.



Type: String

BL_SUPPRESS_NULLIF

Specifies whether to ignore NULLIF conditions defined in the SQL*Loader control file.

***variable***

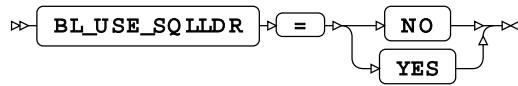
The name of a column (variable), or `_ALL_`.

Specify:

- `variable=YES` to ignore the NULLIF condition for the column *variable*
- `variable=NO` to ignore the NULLIF condition for the column *variable*; this is the default.
- `_ALL_=YES` to ignore NULLIF conditions for all columns,

BL_USE_SQLLDR

Specifies whether bulk inserts are made using Oracle's SQL*Loader.



Note:

SQL*Loader is not included in the Oracle Instant Client and must be installed separately.

NO

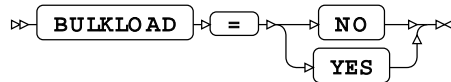
Do not use SQL*Loader for bulk inserts; the Oracle Call Interface (OCI) is used instead.

YES

Use SQL*Loader for bulk inserts.

BULKLOAD

Specifies whether the library connection permits bulk insert into a database table.



Default value: FALSE

NO

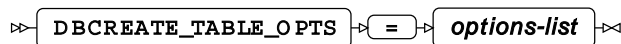
Data cannot be bulk inserted. All other bulkload options are ignored.

YES

Data can be bulk inserted.

DBCREATE_TABLE_OPTS

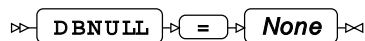
Specifies extra table options to be added to a `CREATE TABLE...` statement after the columns in the table have been defined.



Type: String

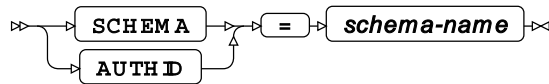
DBNULL

Specifies that a missing value in a dataset is written to a table as a null.



SCHEMA

Specifies the name of the database schema with which the connection interacts.

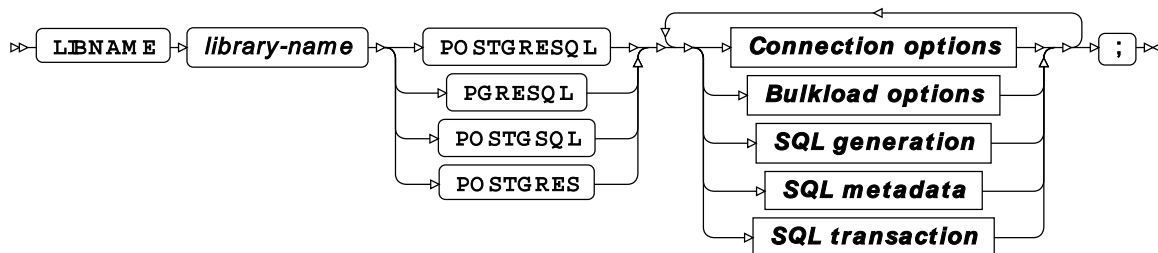


Type: String

The schema is a grouping of database objects and data accessible by the user that can be manipulated through SQL statements.

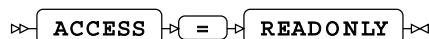
WPS Engine for PostgreSQL

POSTGRESQL

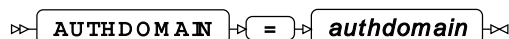


Connection options

ACCESS

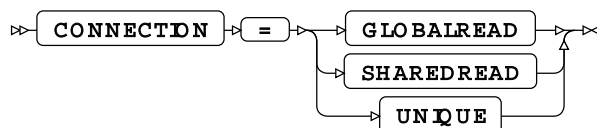


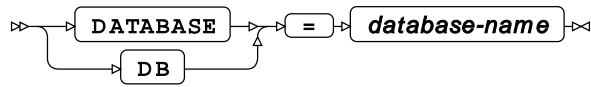
AUTHDOMAIN



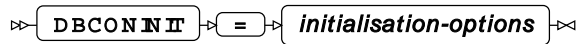
Type: String

CONNECTION

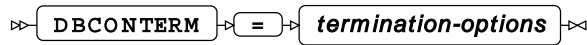


DATABASE

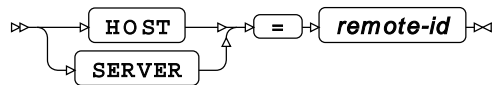
Type: String

DBCONINIT

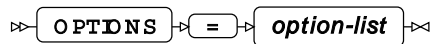
Type: String

DBCONTERM

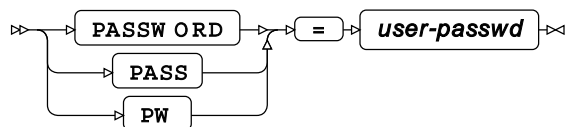
Type: String

HOST

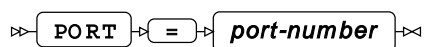
Type: String

OPTIONS

Type: String

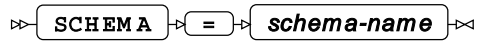
PASSWORD

Type: String

PORT

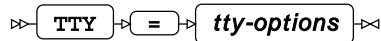
Type: String

SCHEMA



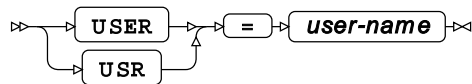
Type: String

TTY



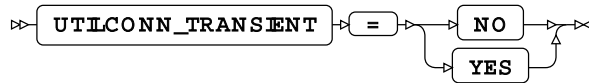
Type: String

USER



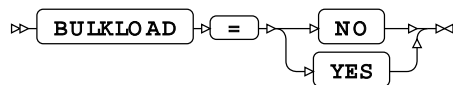
Type: String

UTILCONN_TRANSIENT

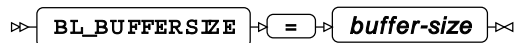


Bulkload options

BULKLOAD

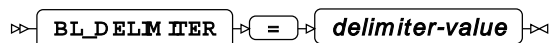


BL_BUFFERSIZE



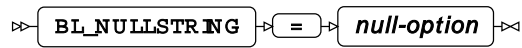
Type: String

BL_DELIMITER



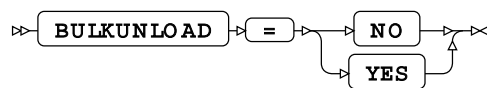
Type: String

BL_NULLSTRING



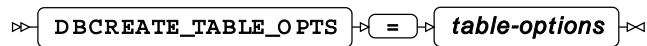
Type: String

BULKUNLOAD



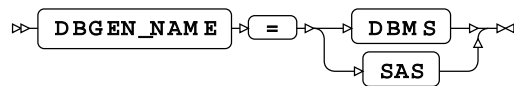
SQL generation

DBCREATE_TABLE_OPTS



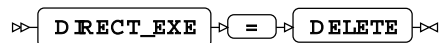
Type: String

DBGEN_NAME

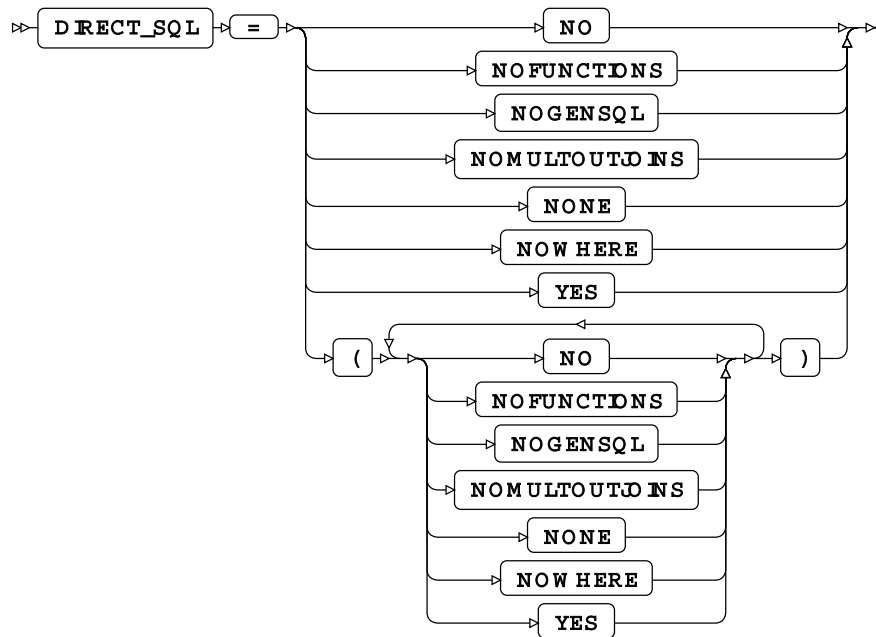


Default value: SAS

DIRECT_EXE

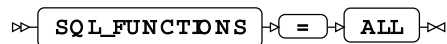


DIRECT_SQL

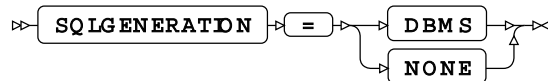


Default value: YES

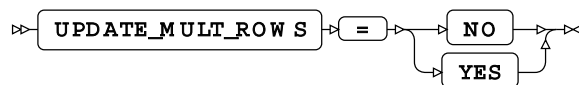
SQL_FUNCTIONS



SQLGENERATION



UPDATE_MULT_ROWS



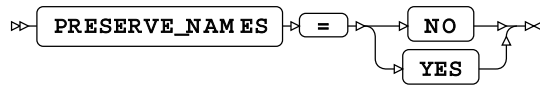
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

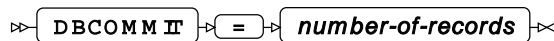
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

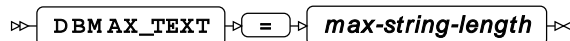
DBCMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

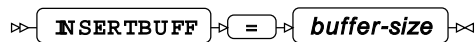


Type: Numeric

Minimum value: 1

Maximum value: 32767

INSERTBUFF

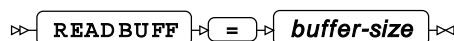


Type: Numeric

Minimum value: 1

Maximum value: 32767

READBUFF

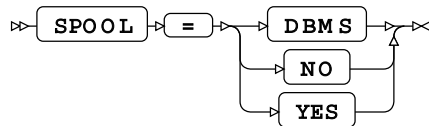


Type: Numeric

Minimum value: 1

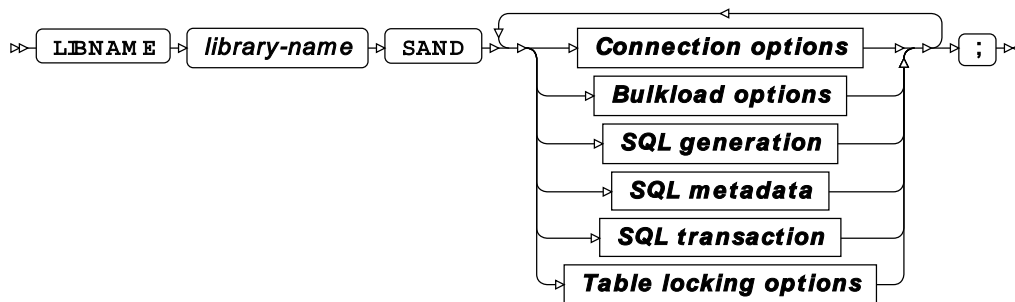
Maximum value: 32767

SPOOL



WPS Engine for Sand

SAND

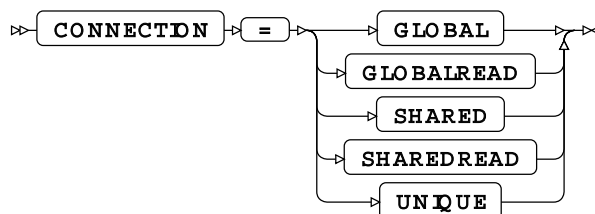


Connection options

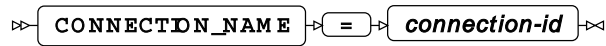
ACCESS



CONNECTION

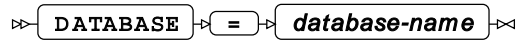


CONNECTION_NAME



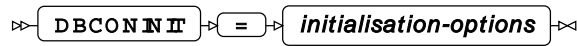
Type: String

DATABASE



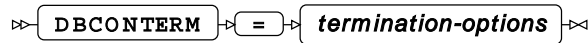
Type: String

DBCONINIT



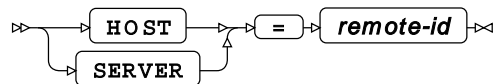
Type: String

DBCONTERM



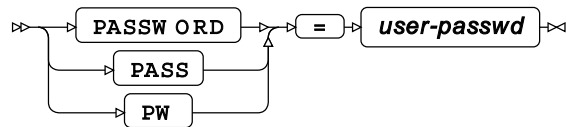
Type: String

HOST



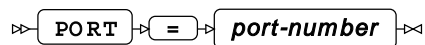
Type: String

PASSWORD



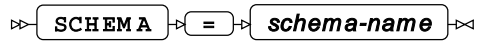
Type: String

PORT



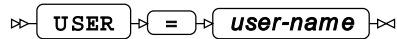
Type: Numeric

SCHEMA



Type: String

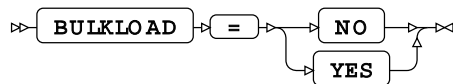
USER



Type: String

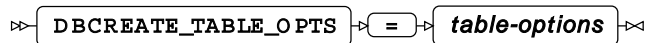
Bulkload options

BULKLOAD



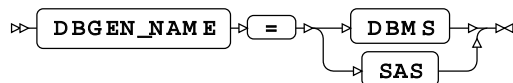
SQL generation

DBCREATE_TABLE_OPTS



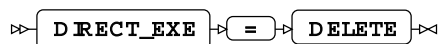
Type: String

DBGEN_NAME

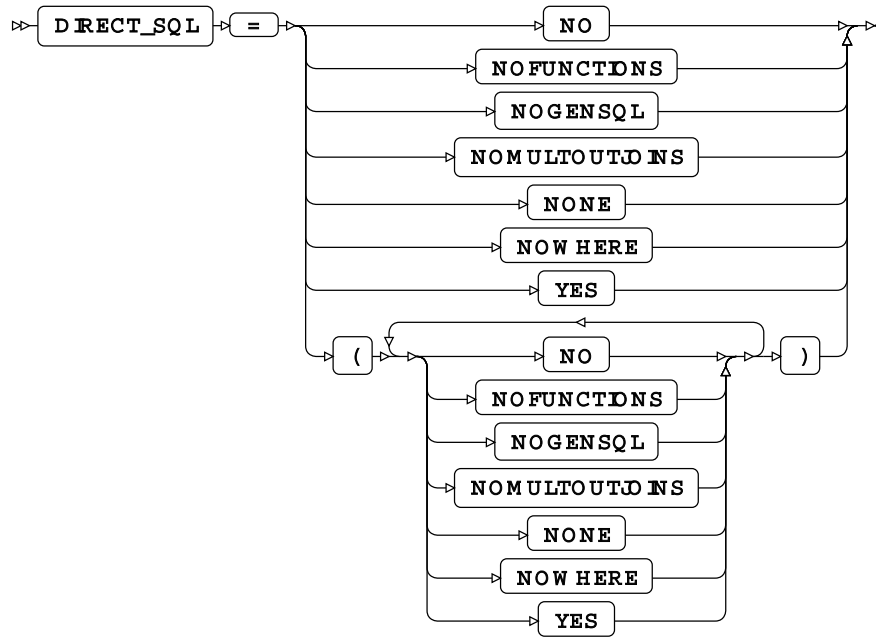


Default value: SAS

DIRECT_EXE



DIRECT_SQL

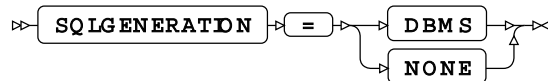


Default value: YES

SQL_FUNCTIONS



SQLGENERATION



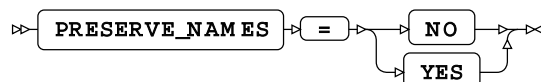
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

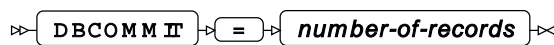
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

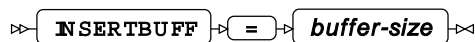
DBCOMMIT



Type: Numeric

Minimum value: 0

INSERTBUFF



Type: Numeric

READBUFF



Type: Numeric

Minimum value: 1

SPOOL

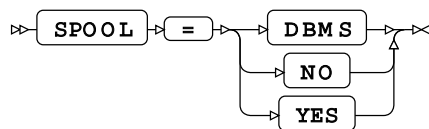
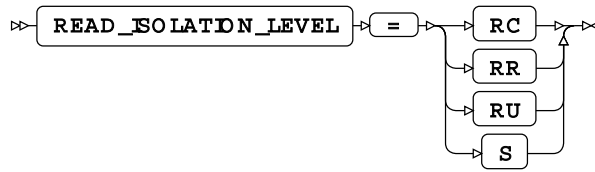
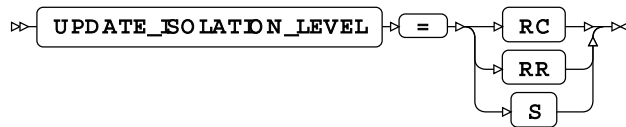


Table locking options

READ_ISOLATION_LEVEL

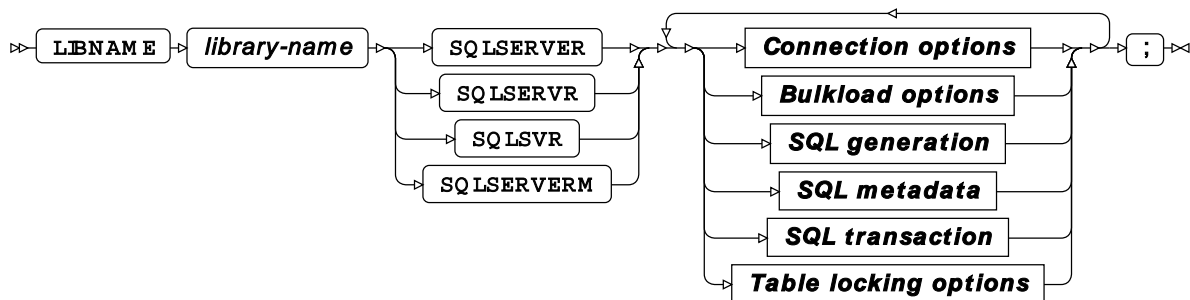


UPDATE_ISOLATION_LEVEL



WPS Engine for SQL Server

SQLSERVER

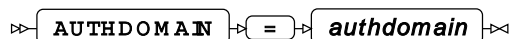


Connection options

ACCESS

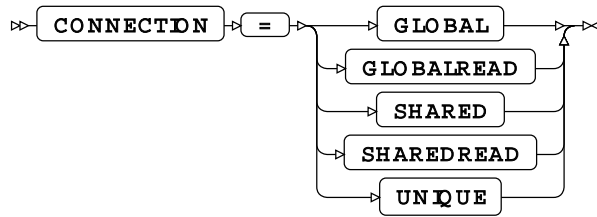


AUTHDOMAIN

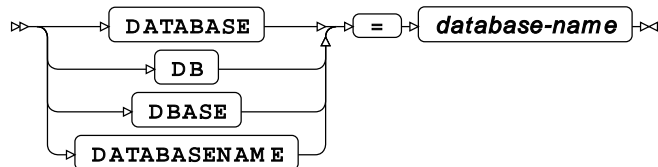


Type: String

CONNECTION

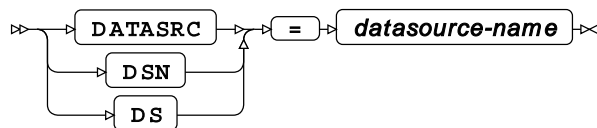


DATABASE



Type: String

DATASRC



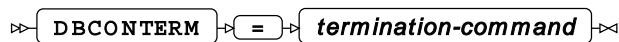
Type: String

DBCONINIT



Type: String

DBCONTERM



Type: String

DBLIBINIT



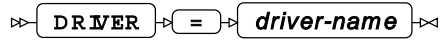
Type: String

DBLIBTERM



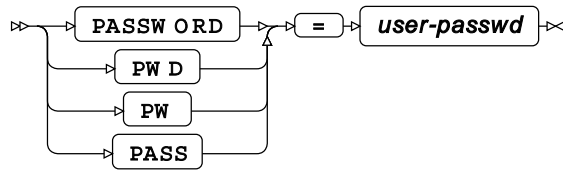
Type: String

DRIVER



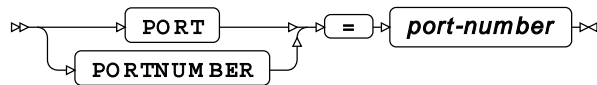
Type: String

PASSWORD



Type: String

PORT

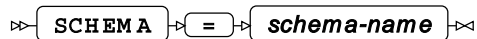


Type: Numeric

Minimum value: 0

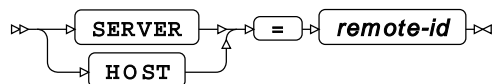
Default value: 1433

SCHEMA



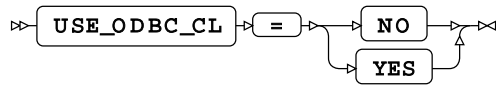
Type: String

SERVER

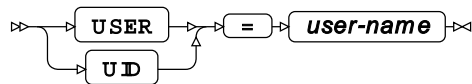


Type: String

USE_ODBC_CL

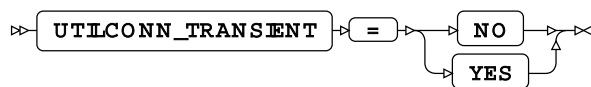


USER



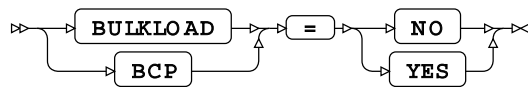
Type: String

UTILCONN_TRANSIENT



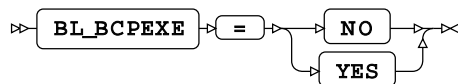
Bulkload options

BULKLOAD



Default value: NO

BL_BCPEXE



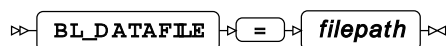
Default value: NO

BL_BCPEXE_PATH



Type: String

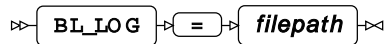
BL_DATAFILE



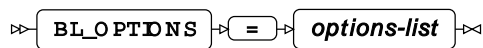
Type: String

BL_DELETE_ONLY_DATAFILE

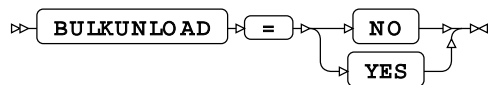
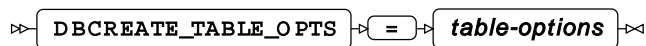
Default value: NO

BL_LOG

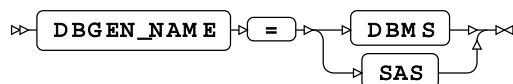
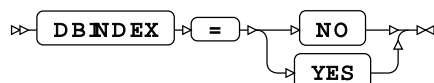
Type: String

BL_OPTIONS

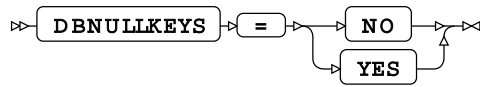
Type: String

BULKUNLOAD**SQL generation****DBCREATE_TABLE_OPTS**

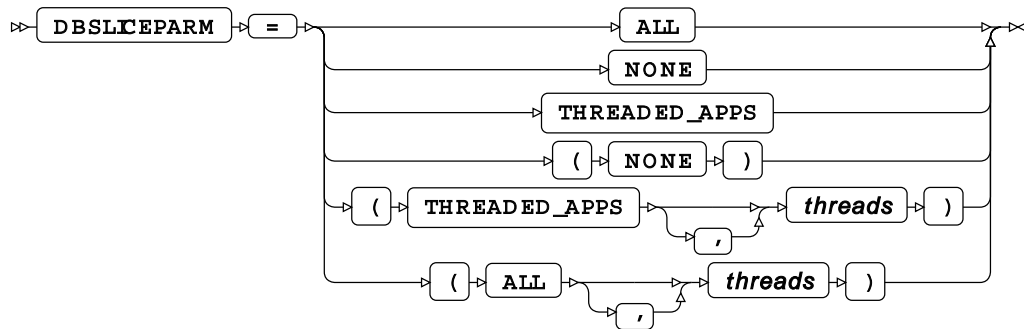
Type: String

DBGEN_NAME**DBINDEX**

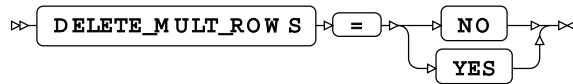
DBNULLKEYS



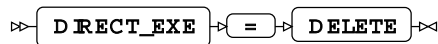
DBSLICEPARM



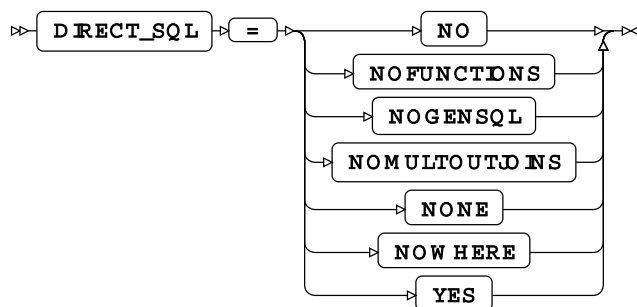
DELETE_MULT_ROWS



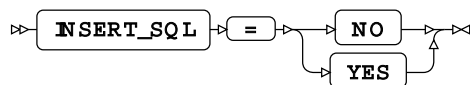
DIRECT_EXE



DIRECT_SQL

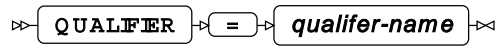


INSERT_SQL



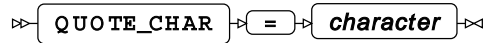
Default value: YES

QUALIFIER



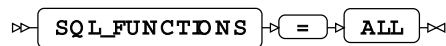
Type: String

QUOTE_CHAR

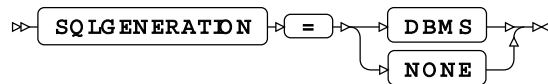


Type: String

SQL_FUNCTIONS

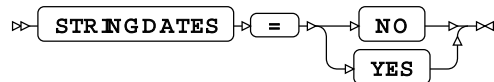


SQLGENERATION

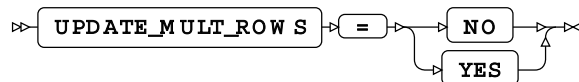


Default value: NONE

STRINGDATES



UPDATE_MULT_ROWS



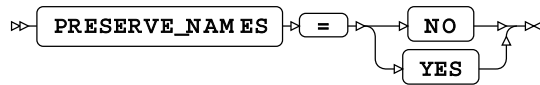
SQL metadata

PRESERVE_COL_NAMES



Default value: YES

PRESERVE_NAMES



Default value: YES

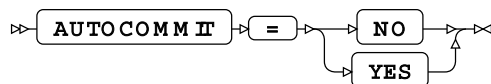
PRESERVE_TAB_NAMES



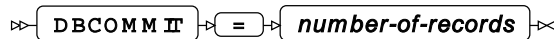
Default value: YES

SQL transaction

AUTOCOMMIT



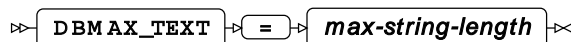
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



Type: Numeric

Minimum value: 1

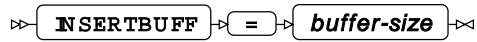
Maximum value: 32767

IGNORE_READ_ONLY_COLUMNS



Default value: NO

INSERTBUFF

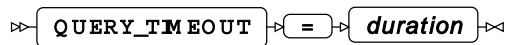


Type: Numeric

Minimum value: 1

Maximum value: 32767

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

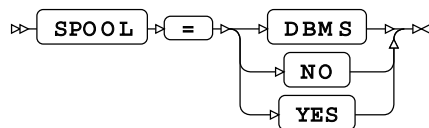


Type: Numeric

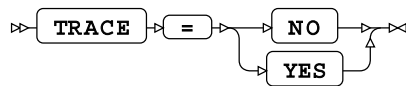
Minimum value: 1

Maximum value: 32767

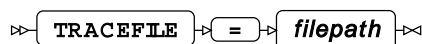
SPOOL



TRACE



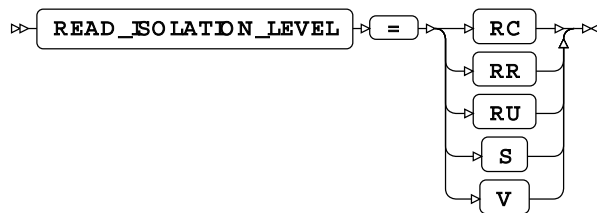
TRACEFILE



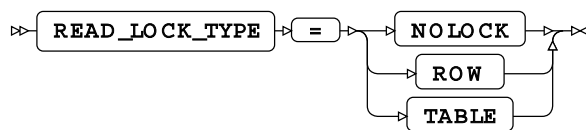
Type: String

Table locking options

READ_ISOLATION_LEVEL

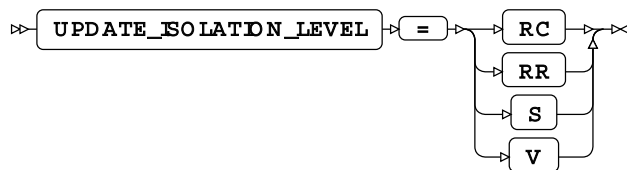


READ_LOCK_TYPE

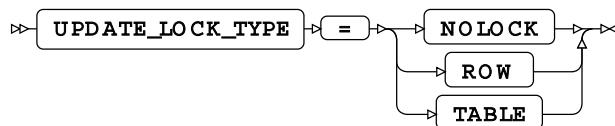


Default value: NOLOCK

UPDATE_ISOLATION_LEVEL



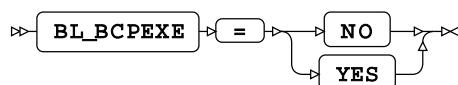
UPDATE_LOCK_TYPE



Default value: NOLOCK

SQLSERVER Dataset Options

BL_BCPEXE



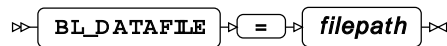
Default value: FALSE

BL_BCPEXE_PATH



Type: String

BL_DATAFILE



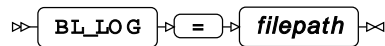
Type: String

BL_DELETE_ONLY_DATAFILE



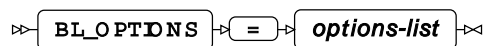
Default value: FALSE

BL_LOG



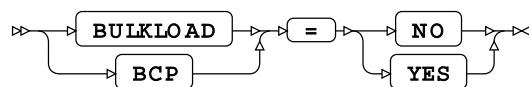
Type: String

BL_OPTIONS



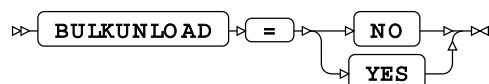
Type: String

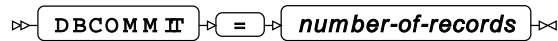
BULKLOAD



Default value: FALSE

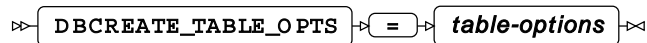
BULKUNLOAD



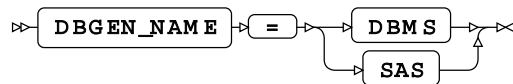
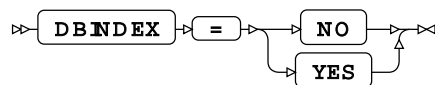
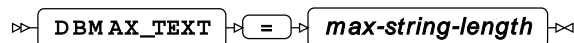
DBCMMIT

Type: Numeric

Minimum value: 0

DBCREATE_TABLE_OPTS

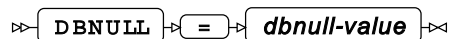
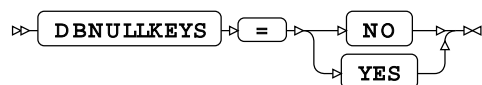
Type: String

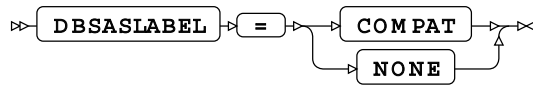
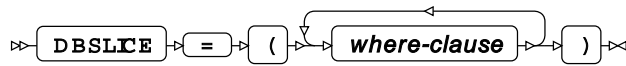
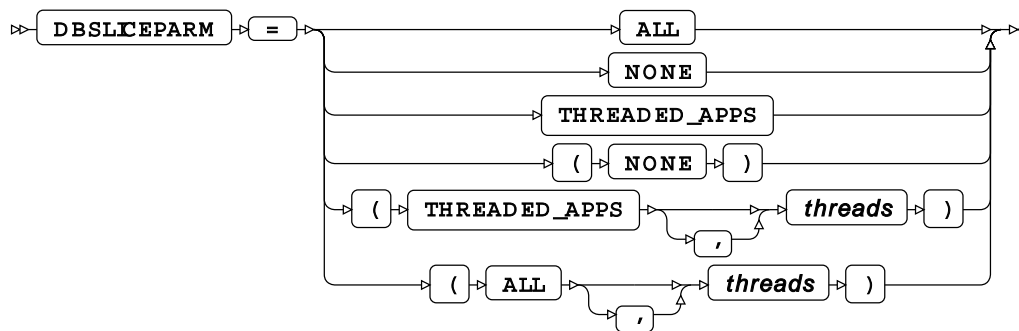
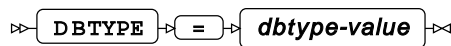
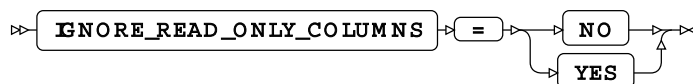
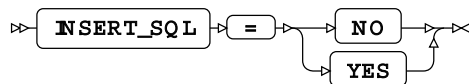
DBGEN_NAME**DBINDEX****DBMAX_TEXT**

Type: Numeric

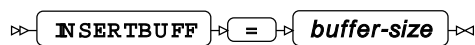
Minimum value: 1

Maximum value: 32767

DBNULL**DBNULLKEYS**

DBSASLABEL**DBSLICE****DBSLICEPARM****DBTYPE****IGNORE_READ_ONLY_COLUMNS****INSERT_SQL**

Default value: TRUE

INSERTBUFF

Type: Numeric

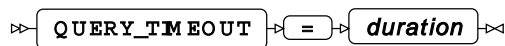
Minimum value: 1

PRESERVE_COL_NAMES



Default value: TRUE

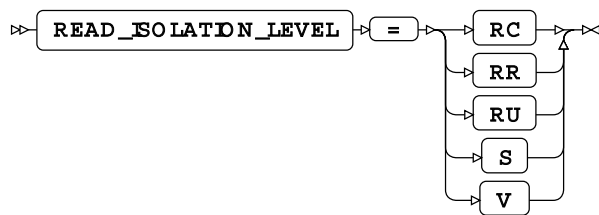
QUERY_TIMEOUT



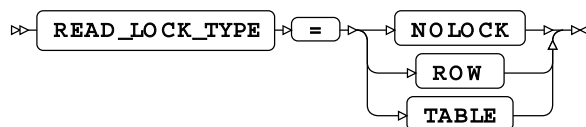
Type: Numeric

Minimum value: 0

READ_ISOLATION_LEVEL



READ_LOCK_TYPE



Default value: NOLOCK

READBUFF

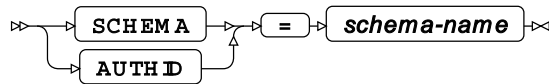


Type: Numeric

Minimum value: 1

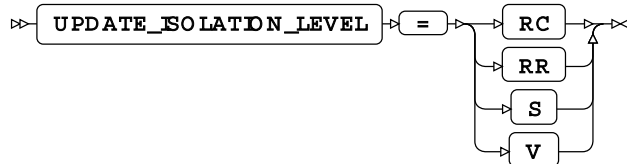
Maximum value: 32767

SCHEMA

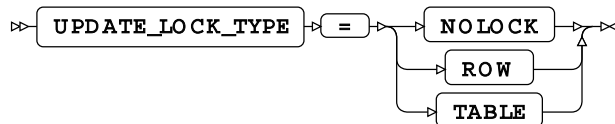


Type: String

UPDATE_ISOLATION_LEVEL

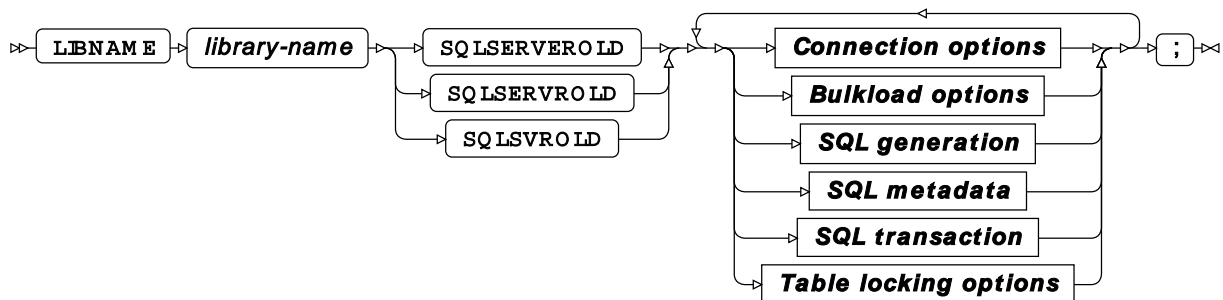


UPDATE_LOCK_TYPE



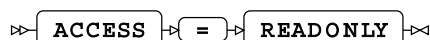
Default value: NOLOCK

SQLSERVEROLD

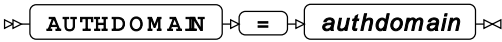


Connection options

ACCESS

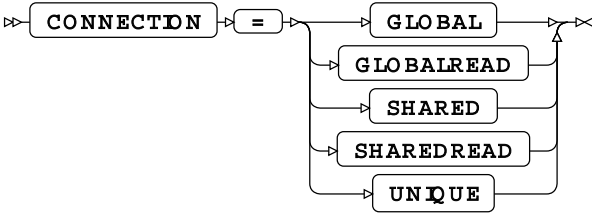


AUTHDOMAIN

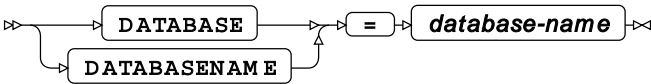


Type: String

CONNECTION

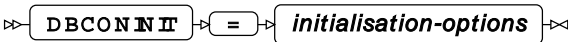


DATABASE



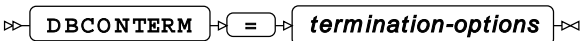
Type: String

DBCONINIT



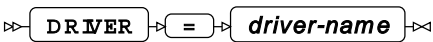
Type: String

DBCONTERM



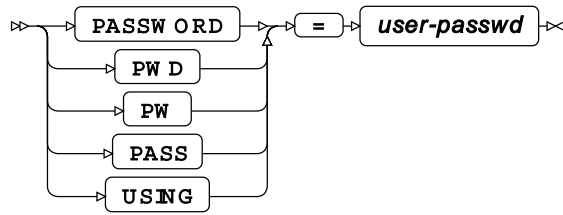
Type: String

DRIVER



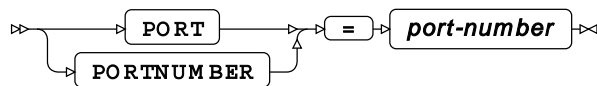
Type: String

PASSWORD



Type: String

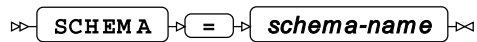
PORT



Type: Numeric

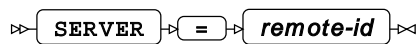
Default value: 1433

SCHEMA



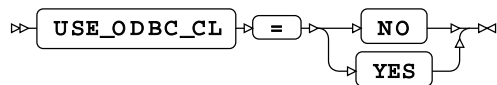
Type: String

SERVER

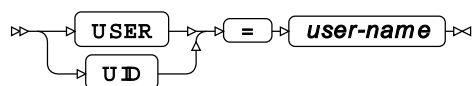


Type: String

USE_ODBC_CL

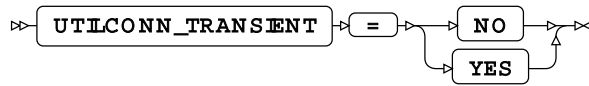


USER



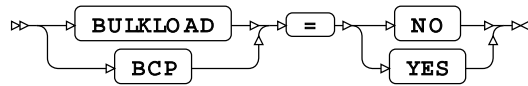
Type: String

UTILCONN_TRANSIENT



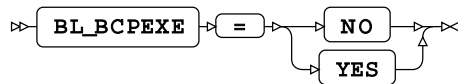
Bulkload options

BULKLOAD



Default value: NO

BL_BCPEXE



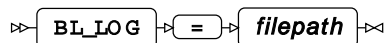
Default value: NO

BL_BCPEXE_PATH



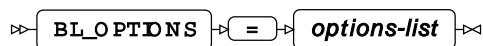
Type: String

BL_LOG



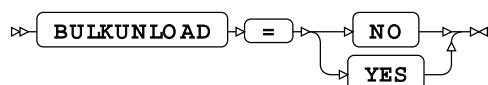
Type: String

BL_OPTIONS



Type: String

BULKUNLOAD



Default value: NO

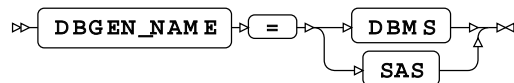
SQL generation

DBCREATE_TABLE_OPTS



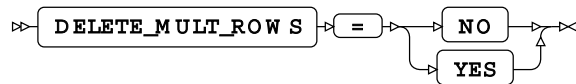
Type: String

DBGEN_NAME

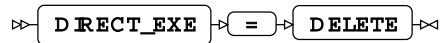


Default value: SAS

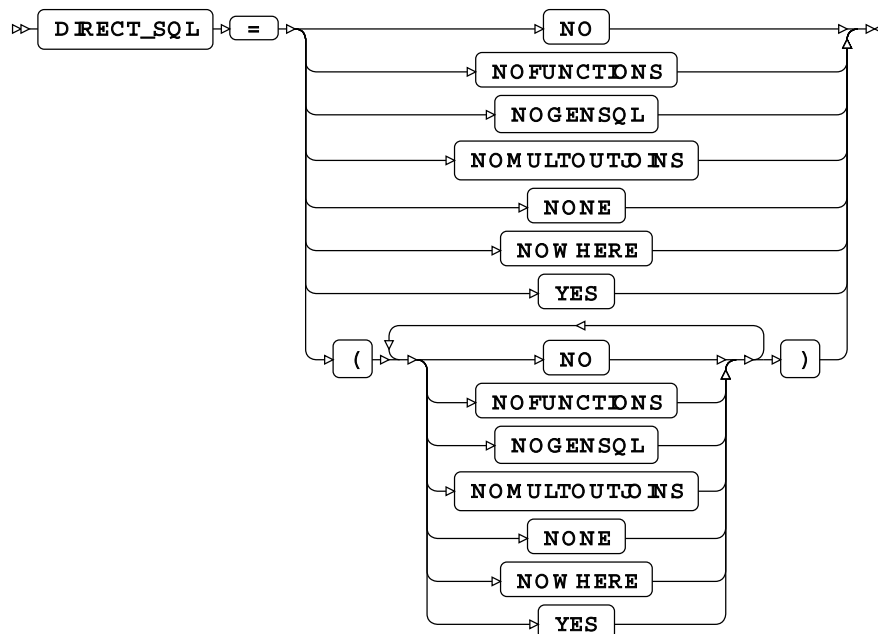
DELETE_MULT_ROWS



DIRECT_EXE

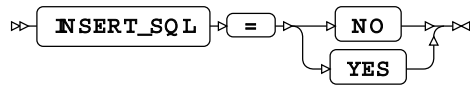


DIRECT_SQL

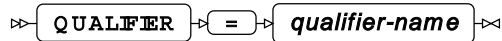


Default value: YES

INSERT_SQL

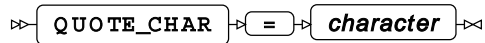


QUALIFIER



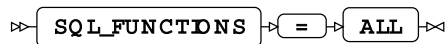
Type: String

QUOTE_CHAR

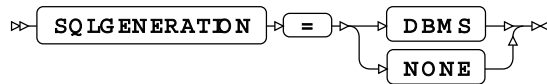


Type: String

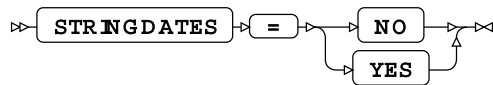
SQL_FUNCTIONS



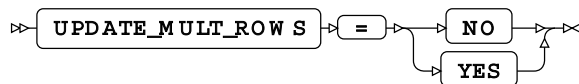
SQLGENERATION



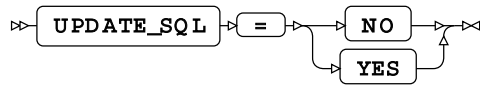
STRINGDATES



UPDATE_MULT_ROWS

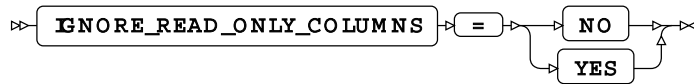


UPDATE_SQL



SQL metadata

IGNORE_READ_ONLY_COLUMNS



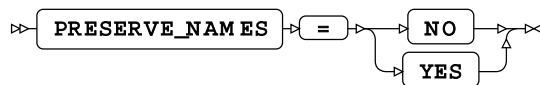
Default value: NO

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

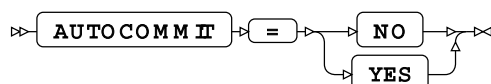
PRESERVE_TAB_NAMES



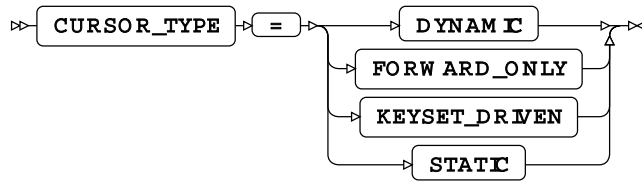
Default value: NO

SQL transaction

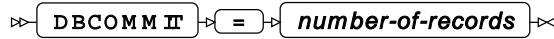
AUTOCOMMIT



CURSOR_TYPE



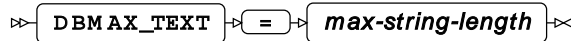
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



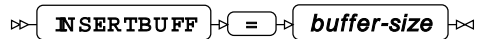
Type: Numeric

Minimum value: 1

Maximum value: 32767

Default value: 4000

INSERTBUFF

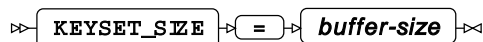


Type: Numeric

Minimum value: 1

Maximum value: 32767

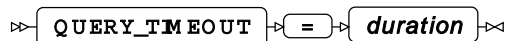
KEYSET_SIZE



Type: Numeric

Minimum value: 0

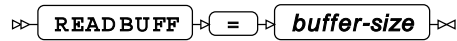
QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

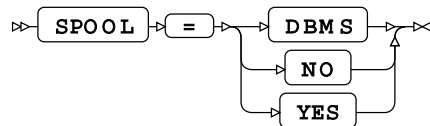


Type: Numeric

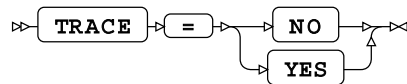
Minimum value: 1

Maximum value: 32767

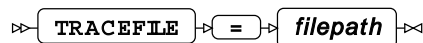
SPOOL



TRACE



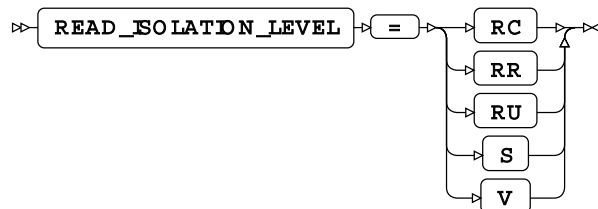
TRACEFILE



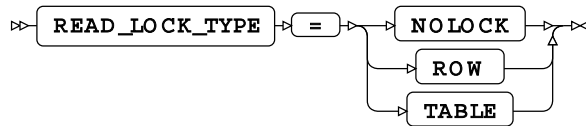
Type: String

Table locking options

READ_ISOLATION_LEVEL

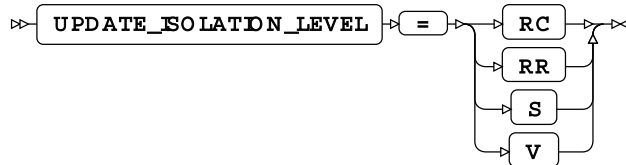


READ_LOCK_TYPE

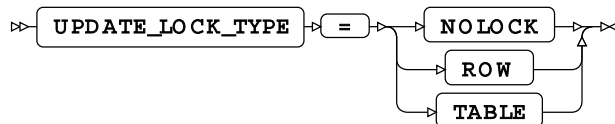


Default value: NOLOCK

UPDATE_ISOLATION_LEVEL



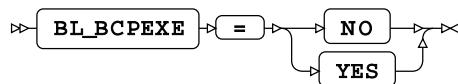
UPDATE_LOCK_TYPE



Default value: NOLOCK

SQLSERVEROLD Dataset Options

BL_BCPEXE



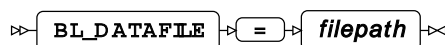
Default value: FALSE

BL_BCPEXE_PATH



Type: String

BL_DATAFILE



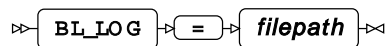
Type: String

BL_DELETE_ONLY_DATAFILE



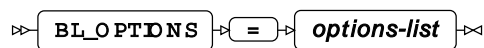
Default value: FALSE

BL_LOG



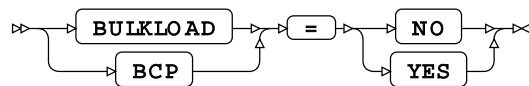
Type: String

BL_OPTIONS



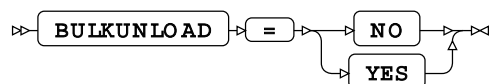
Type: String

BULKLOAD



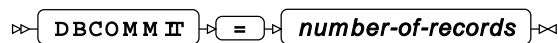
Default value: FALSE

BULKUNLOAD



Default value: FALSE

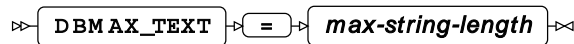
DBCMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT



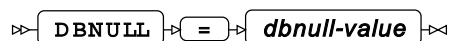
Type: Numeric

Minimum value: 1

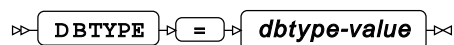
Maximum value: 32767

Default value: 4000

DBNULL



DBTYPE

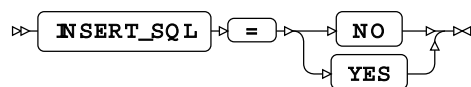


IGNORE_READ_ONLY_COLUMNS

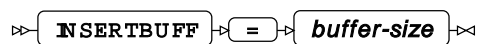


Default value: FALSE

INSERT_SQL



INSERTBUFF

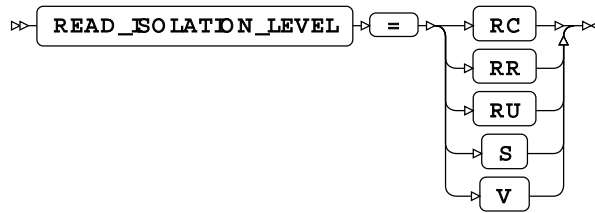


Type: Numeric

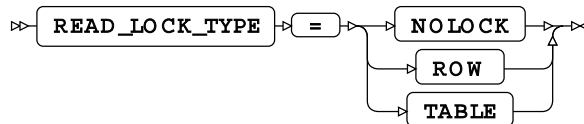
Minimum value: 1

Maximum value: 32767

READ_ISOLATION_LEVEL



READ_LOCK_TYPE



Default value: NOLOCK

READBUFF

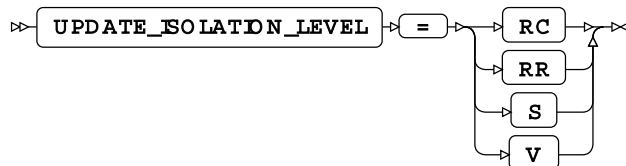


Type: Numeric

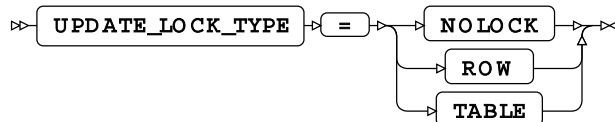
Minimum value: 1

Maximum value: 32767

UPDATE_ISOLATION_LEVEL



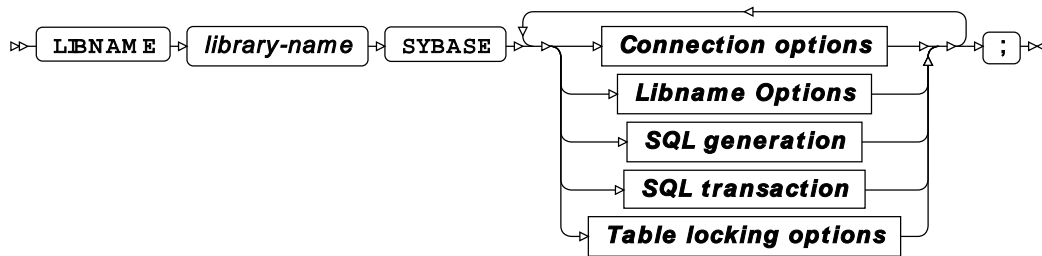
UPDATE_LOCK_TYPE



Default value: NOLOCK

WPS Engine for Sybase

SYBASE

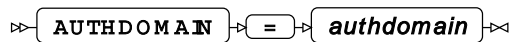


Connection options

ACCESS

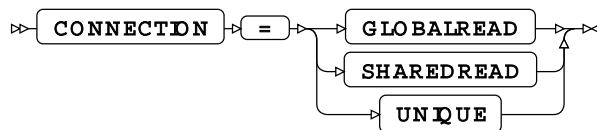


AUTHDOMAIN

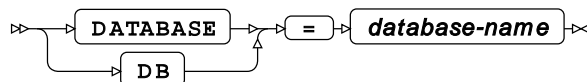


Type: String

CONNECTION

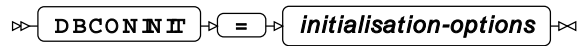


DATABASE



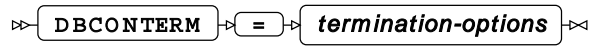
Type: String

DBCONINIT



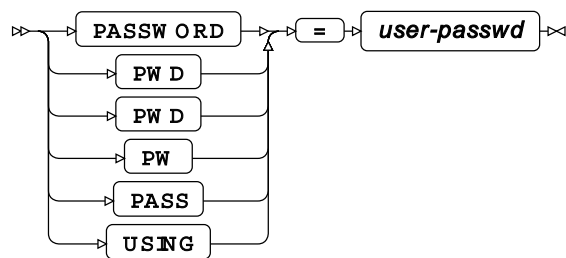
Type: String

DBCONTERM



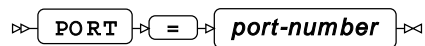
Type: String

PASSWORD



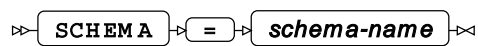
Type: String

PORT



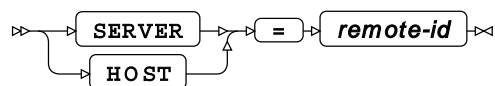
Type: String

SCHEMA



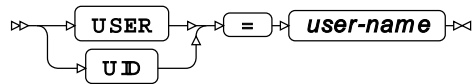
Type: String

SERVER



Type: String

USER



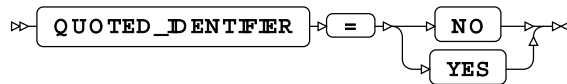
Type: String

UTILCONN_TRANSIENT



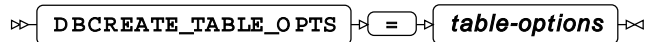
Libname Options

QUOTED_IDENTIFIER



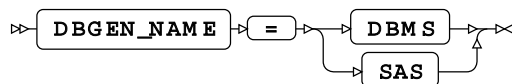
SQL generation

DBCREATE_TABLE_OPTS



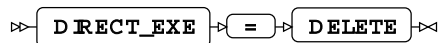
Type: String

DBGEN_NAME

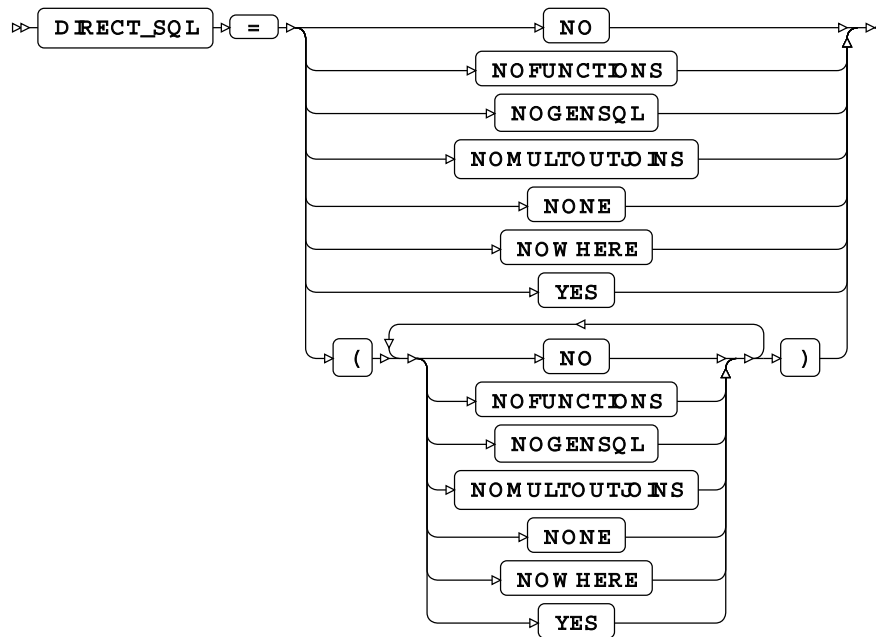


Default value: SAS

DIRECT_EXE

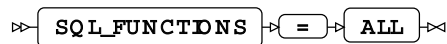


DIRECT_SQL

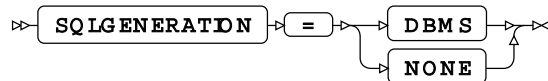


Default value: YES

SQL_FUNCTIONS

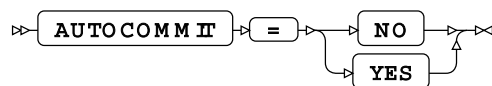


SQLGENERATION

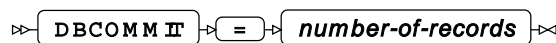


SQL transaction

AUTOCOMMIT

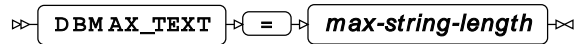


DBCOMMIT



Type: Numeric

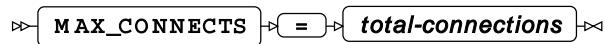
Minimum value: 0

DBMAX_TEXT

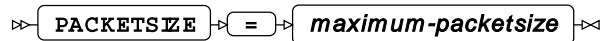
Type: Numeric

Minimum value: 1

Maximum value: 32767

MAX_CONNECTS

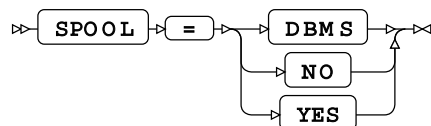
Type: Numeric

PACKETSIZE

Type: Numeric

READBUFF

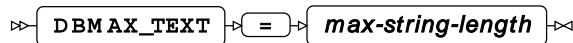
Type: Numeric

SPOOL**Table locking options****READ_LOCK_TYPE****UPDATE_ISOLATION_LEVEL**

Type: Numeric

SYBASE Dataset Options

DBMAX_TEXT



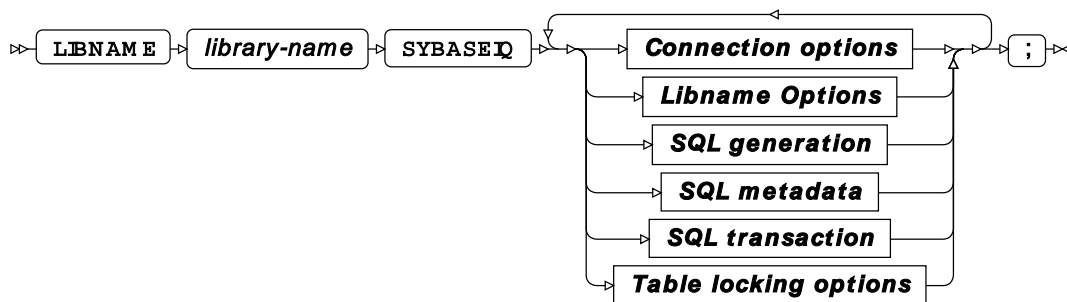
Type: Numeric

Minimum value: 1

Maximum value: 32767

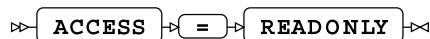
WPS Engine for Sybase IQ

SYBASEIQ

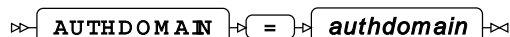


Connection options

ACCESS

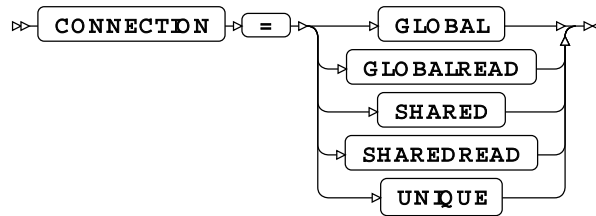


AUTHDOMAIN

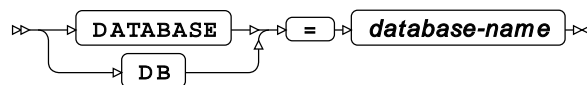


Type: String

CONNECTION

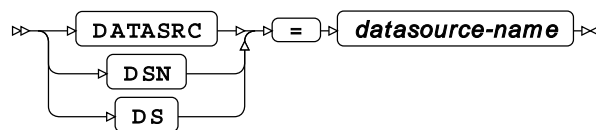


DATABASE



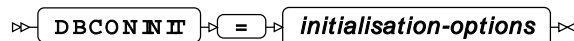
Type: String

DATASRC



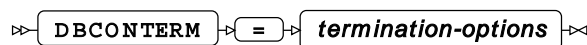
Type: String

DBCONINIT



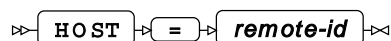
Type: String

DBCONTERM



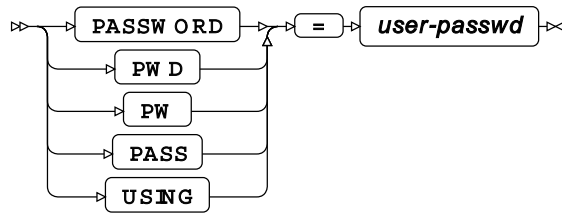
Type: String

HOST



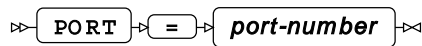
Type: String

PASSWORD



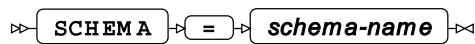
Type: String

PORT



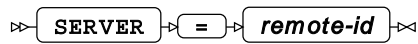
Type: String

SCHEMA



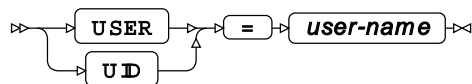
Type: String

SERVER



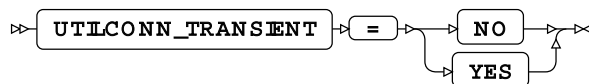
Type: String

USER



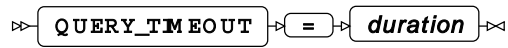
Type: String

UTILCONN_TRANSIENT



Libname Options

QUERY_TIMEOUT

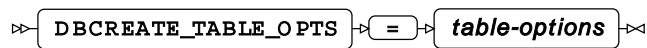


Type: Numeric

Minimum value: 0

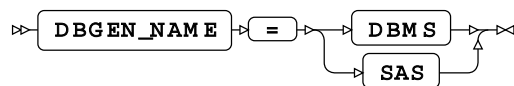
SQL generation

DBCREATE_TABLE_OPTS



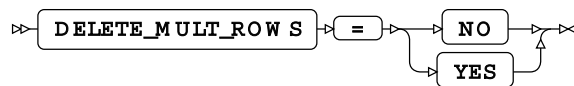
Type: String

DBGEN_NAME

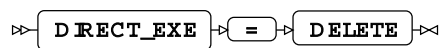


Default value: SAS

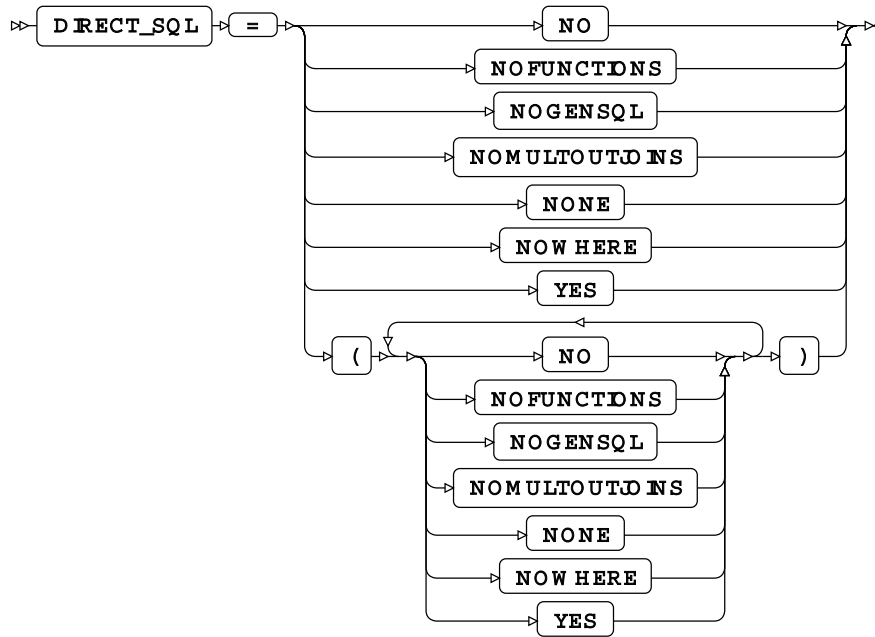
DELETE_MULT_ROWS



DIRECT_EXE

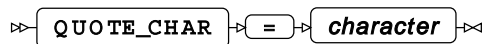


DIRECT_SQL



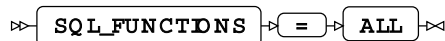
Default value: YES

QUOTE_CHAR

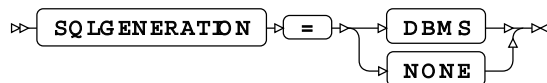


Type: String

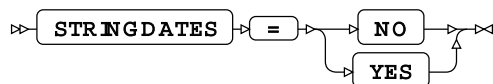
SQL_FUNCTIONS



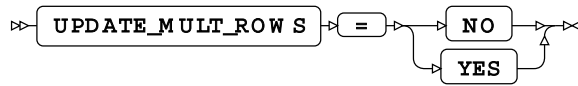
SQLGENERATION



STRINGDATES

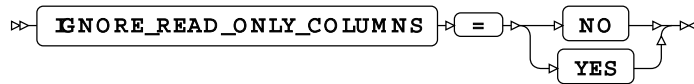


UPDATE_MULT_ROWS



SQL metadata

IGNORE_READ_ONLY_COLUMNS



PRESERVE_COL_NAMES



Default value: NO

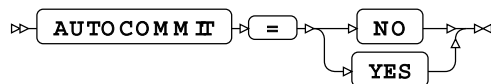
PRESERVE_TAB_NAMES



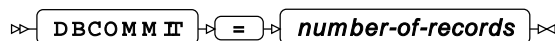
Default value: NO

SQL transaction

AUTOCOMMIT



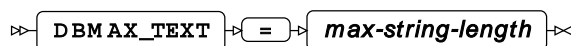
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

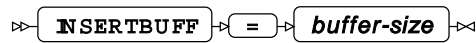


Type: Numeric

Minimum value: 1

Maximum value: 32767

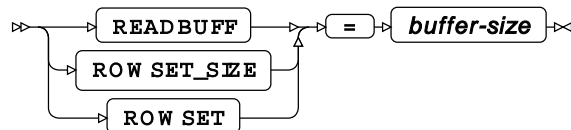
INSERTBUFF



Type: Numeric

Minimum value: 1

READBUFF

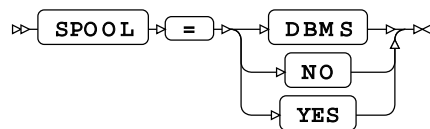


Type: Numeric

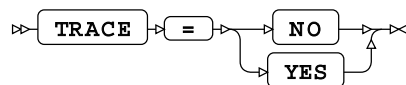
Minimum value: 1

Maximum value: 65535

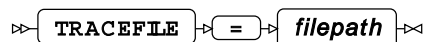
SPOOL



TRACE



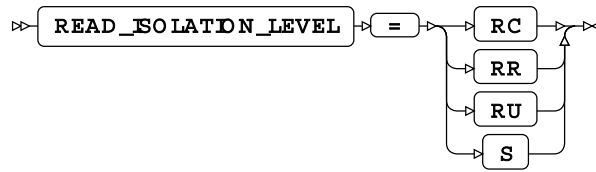
TRACEFILE



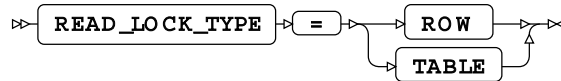
Type: String

Table locking options

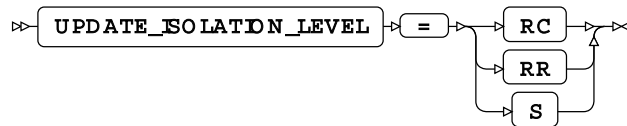
READ_ISOLATION_LEVEL



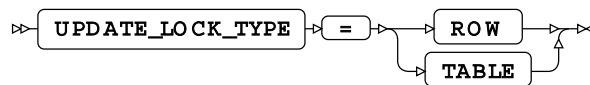
READ_LOCK_TYPE



UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE



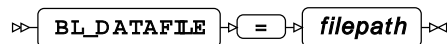
SYBASEIQ Dataset Options

BL_CLIENT_DATAFILE



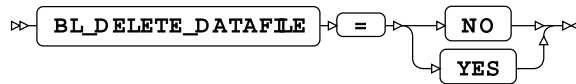
Type: String

BL_DATAFILE

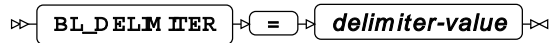


Type: String

BL_DELETE_DATAFILE

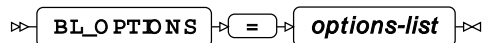


BL_DELIMITER



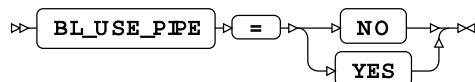
Type: String

BL_OPTIONS

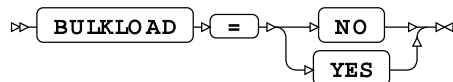


Type: String

BL_USE_PIPE

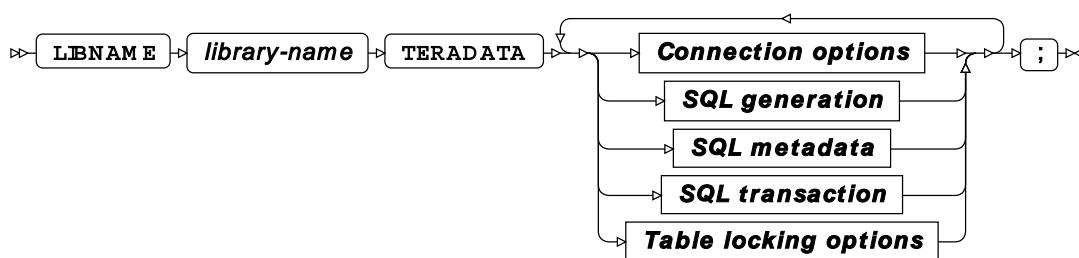


BULKLOAD



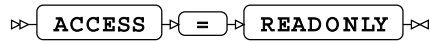
WPS Engine for Teradata

TERADATA

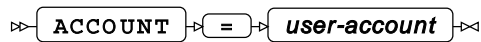


Connection options

ACCESS

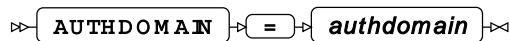


ACCOUNT



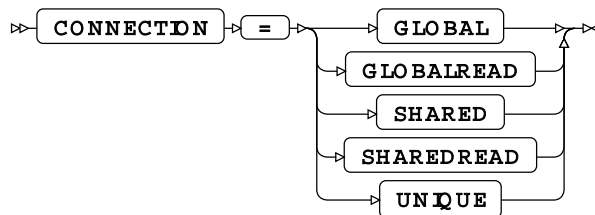
Type: String

AUTHDOMAIN

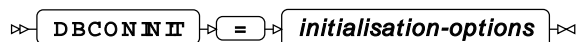


Type: String

CONNECTION

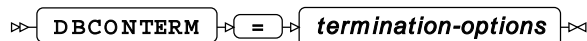


DBCONINIT



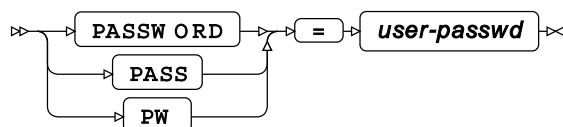
Type: String

DBCONTERM



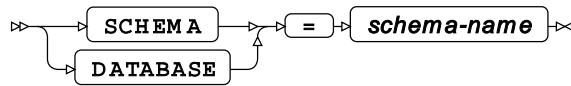
Type: String

PASSWORD



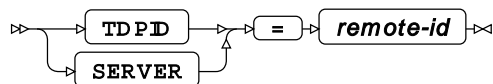
Type: String

SCHEMA



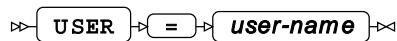
Type: String

TDPID



Type: String

USER



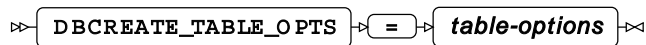
Type: String

UTILCONN_TRANSIENT



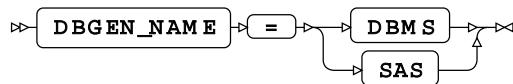
SQL generation

DBCREATE_TABLE_OPTS



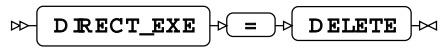
Type: String

DBGEN_NAME

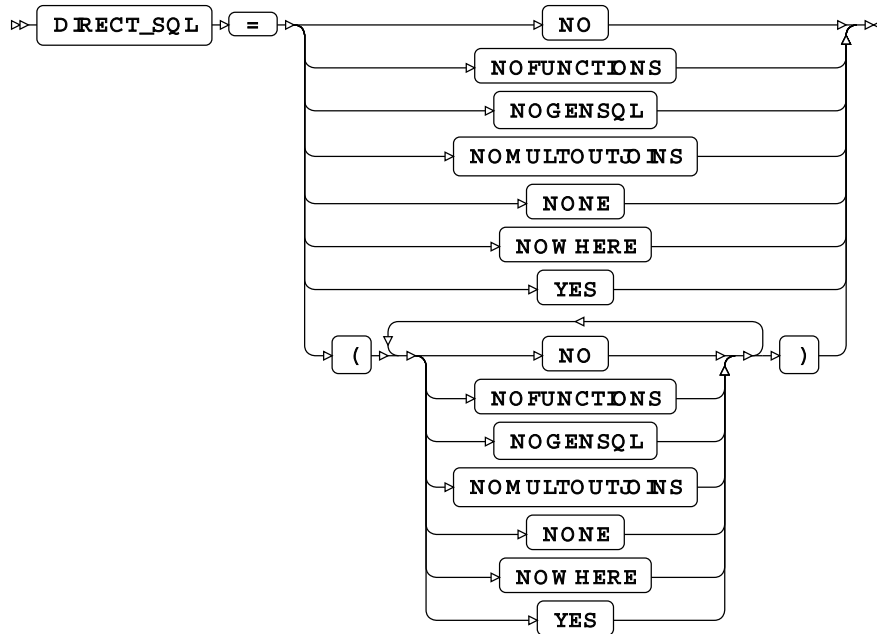


Default value: SAS

DIRECT_EXE

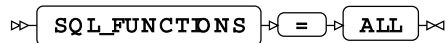


DIRECT_SQL

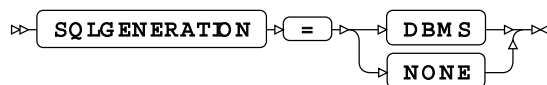


Default value: YES

SQL_FUNCTIONS



SQLGENERATION



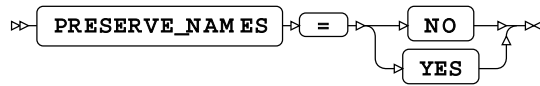
SQL metadata

PRESERVE_COL_NAMES



Default value: NO

PRESERVE_NAMES



Default value: NO

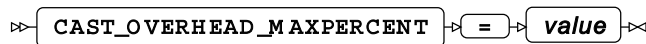
PRESERVE_TAB_NAMES



Default value: NO

SQL transaction

CAST_OVERHEAD_MAXPERCENT

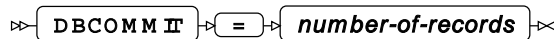


Type: Numeric

Minimum value: 0

Maximum value: 100

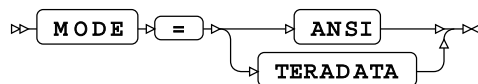
DBCOMMIT



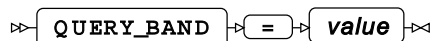
Type: Numeric

Minimum value: 0

MODE



QUERY_BAND



Type: String

SPOOL

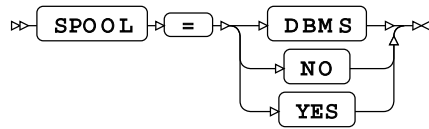
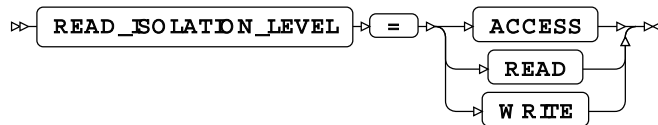
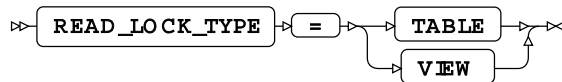


Table locking options

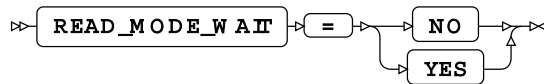
READ_ISOLATION_LEVEL



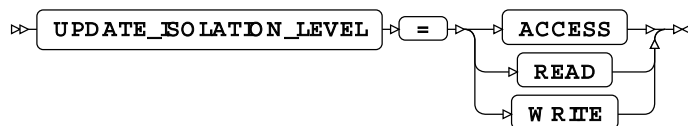
READ_LOCK_TYPE



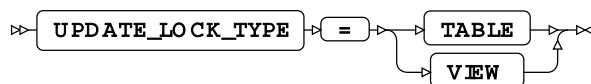
READ_MODE_WAIT



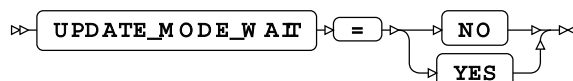
UPDATE_ISOLATION_LEVEL



UPDATE_LOCK_TYPE

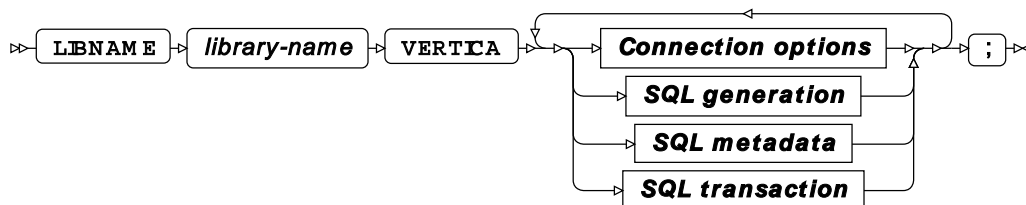


UPDATE_MODE_WAIT



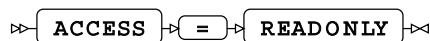
WPS Engine for Vertica

VERTICA



Connection options

ACCESS

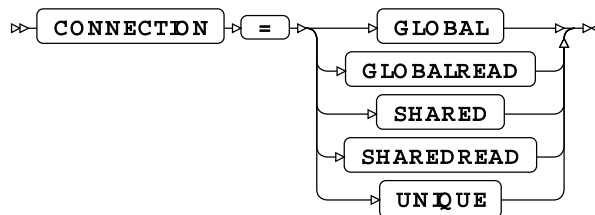


AUTHDOMAIN

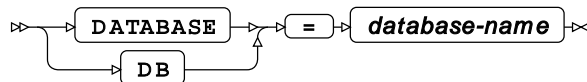


Type: String

CONNECTION

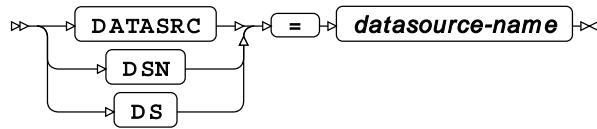


DATABASE



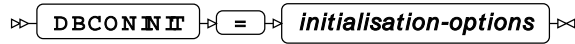
Type: String

DATASRC



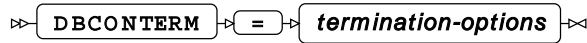
Type: String

DBCONINIT



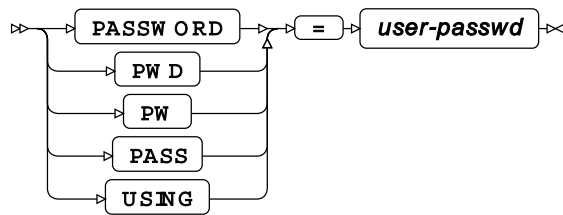
Type: String

DBCONTERM



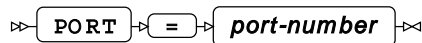
Type: String

PASSWORD



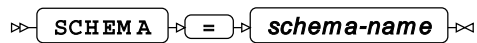
Type: String

PORT



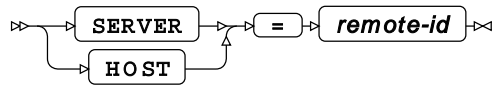
Type: String

SCHEMA



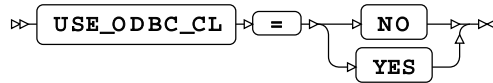
Type: String

SERVER

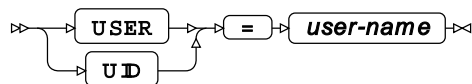


Type: String

USE_ODBC_CL

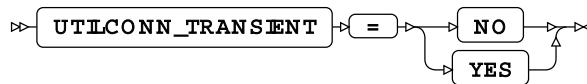


USER



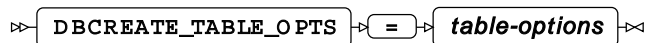
Type: String

UTILCONN_TRANSIENT



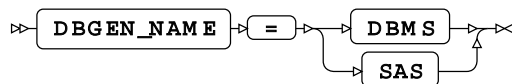
SQL generation

DBCREATE_TABLE_OPTS



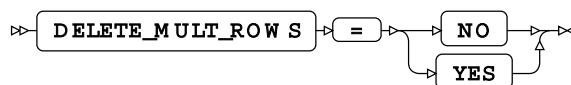
Type: String

DBGEN_NAME

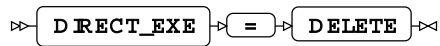


Default value: SAS

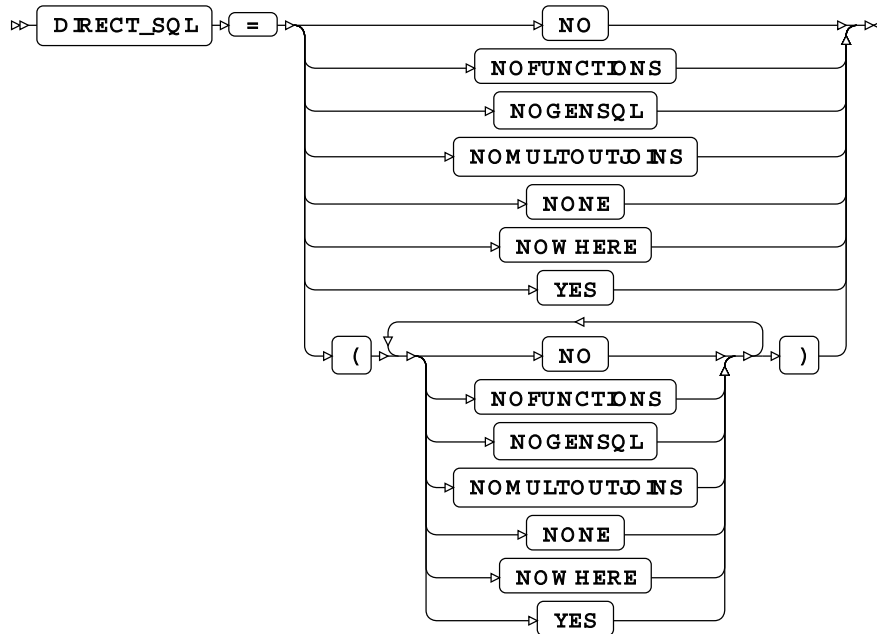
DELETE_MULT_ROWS



DIRECT_EXE

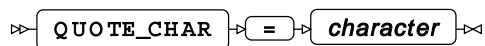


DIRECT_SQL



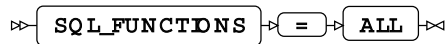
Default value: YES

QUOTE_CHAR

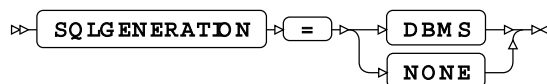


Type: String

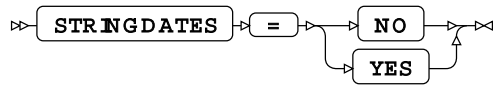
SQL_FUNCTIONS



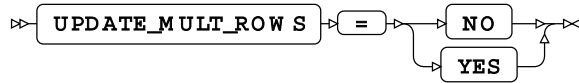
SQLGENERATION



STRINGDATES

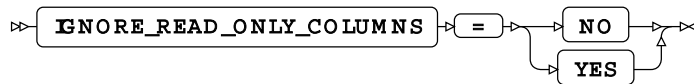


UPDATE_MULT_ROWS



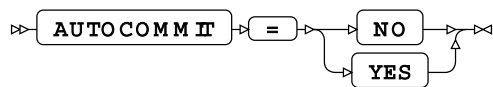
SQL metadata

IGNORE_READ_ONLY_COLUMNS

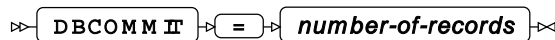


SQL transaction

AUTOCOMMIT



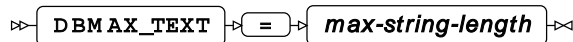
DBCOMMIT



Type: Numeric

Minimum value: 0

DBMAX_TEXT

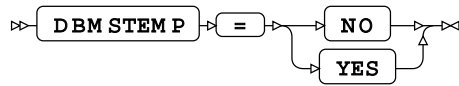


Type: Numeric

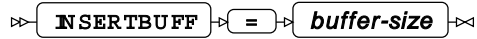
Minimum value: 1

Maximum value: 32767

DBMSTEMP



INSERTBUFF

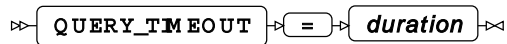


Type: Numeric

Minimum value: 1

Maximum value: 32767

QUERY_TIMEOUT



Type: Numeric

Minimum value: 0

READBUFF

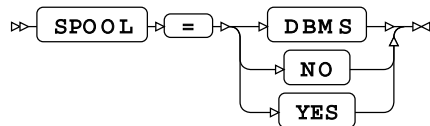


Type: Numeric

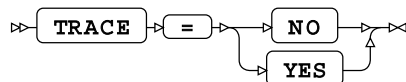
Minimum value: 1

Maximum value: 32767

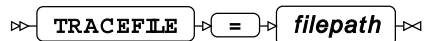
SPOOL



TRACE



TRACEFILE



Type: String

Legal Notices

(c) 2022 World Programming

This information is confidential and subject to copyright. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system.

Trademarks

WPS and World Programming are registered trademarks or trademarks of World Programming Limited in the European Union and other countries. (r) or ® indicates a Community trademark.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

All other trademarks are the property of their respective owner.

General Notices

World Programming Limited is not associated in any way with the SAS Institute.

WPS is not the SAS System.

The phrases "SAS", "SAS language", and "language of SAS" used in this document are used to refer to the computer programming language often referred to in any of these ways.

The phrases "program", "SAS program", and "SAS language program" used in this document are used to refer to programs written in the SAS language. These may also be referred to as "scripts", "SAS scripts", or "SAS language scripts".

The phrases "IML", "IML language", "IML syntax", "Interactive Matrix Language", and "language of IML" used in this document are used to refer to the computer programming language often referred to in any of these ways.

WPS includes software developed by third parties. More information can be found in the THANKS or acknowledgments.txt file included in the WPS installation.